Пояснювальна записка
до дипломного проекту
на тему:
«Апаратна реалізація генетичного алгоритму для навчання нейронних мереж»

Київ – 2019 рік

ЗМІСТ

ВСТУП

Розвиток будь-якого іншого напрямку науки як і розвиток теорії автоматичного управління, характеризується підвищенням складності вирішуваних задач і підвищенням якісних показників необхідних рішень. В той час як реальні об’єкти є за своєю природою нелінійними, традиційні методи управління в основному опираються на теорію лінійних систем. До класу нелінійних динамічних систем відносяться нейромережеві системи управління, які являють собою новий високотехнологічний напрямок в теорії управління.[1]

За рахунок розпаралелювання вхідної інформації досягається висока швидкодія, а здатність до навчання нейронних мереж робить цю технологію вельми привабливою для створення пристроїв управління в автоматичних системах.

На даний момент основними методами реалізації нейромережевих систем управління є програми, з використанням комп'ютерної техніки чи спеціалізованих контролерів, побудованих на її основі, що значно звужує коло практичної реалізації систем управління через значну вартість таких регуляторів і робить їх практично недоступними для використання в простих системах керування, окрім того, комп'ютерні нейромережеві регулятори мають обмежену швидкодію та потребують значних затрат часу на навчання.

Рекурентність і послідовність дій процедури навчання нейромережі при її реалізації на всій множині налагоджувальних параметрів не дозволяє повністю вирішити проблему швидкодії процедури навчання нейромережевих структур в темпі з динамікою об'єкта управління.[2]

Єдиною альтернативою цьому є розпаралелення процедури навчання та роботи внутрішніх елементів нейромережевих структур. Такі можливості з'являються при апаратно-програмній реалізації нейромережевих структур побудованих на програмованих логічних інтегрованих структурах (ПЛІС-FPGA).

Метою роботи є розробка апаратної реалізації генетичного алгоритму на ПЛІС з використанням мови опису апаратури інтегральних схем VHDL простих нейронних мереж(одношарового перцептрону), що дозволить значно підвищити швидкодію їх роботи, за рахунок паралельних обчислень з мінімальним використанням обчислювального ресурсу.

Актуальність роботи пов’язана з наростаючим інтересом використання генетичних алгоритмів в автономних системах, питаннями збільшення швидкодії генетичних алгоритмів, що потребують багато часу, і розвитком нового напрямку в області проектування апаратних засобів, які отримали назву еволюційні апаратні засоби.

Як результат розширення області використання, найбільш гостро постає питання розробки методів підвищення ефективності генетичних алгоритмів з цілого набору критеріїв. Вирішенням подібних питань виступає апаратна реалізація генетичних алгоритмів.

ПЕРЕЛІК СКОРОЧЕНЬ

ШНМ – штучна нейронна мережа

ДНК – дезоксирибо нуклеїнова кислота

ПЛІС – програмована логічна інтегральна схема

FPGA(англ. Field-Programmable Gate Array) – програмована користувачем вентильна матриця, ПКВМ

VHDL (англ. VHSIC (Very high speed integrated circuits) Hardware Description Language) – мова опису апаратури інтегральних схем

LUTs (Look Up Table) – вентилі логічної матриці

ГА – генетичний алгоритм

БНМ – багатошарова нейронна мережа

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Штучний нейрон та штучні нейронні мережі

Штучним нейроном називають вузол штучної нейронної мережі. Модель штучного нейрона є прототипом природного нейрона. Нейрон приймає на вхід певну множину сигналів, які є виходом інших нейронів. З'єднуючи виходи одних нейронів з входами інших, штучні нейрони об'єднуються в мережі.[1] На рисунку 1.1 зображена модель штучного нейрона з трьома входами (дендритами), причому синапси цих дендритів мають ваги $w_1,\ w_2,\ w_3$.

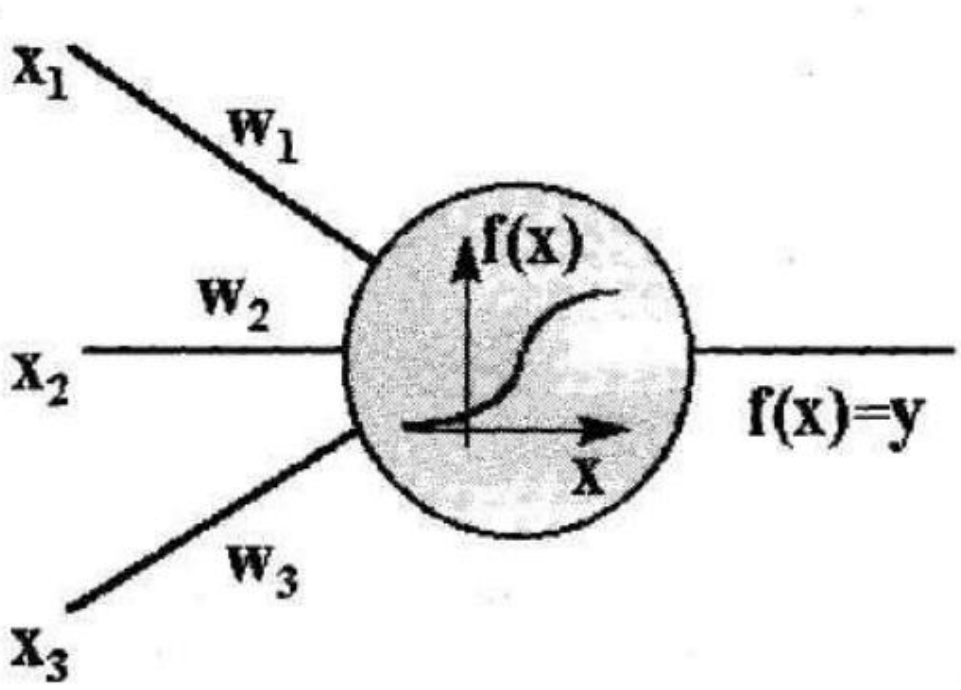

Рисунок 1.1 – Модель штучного нейрона[23]

Нехай до синапсів надходять імпульси сили $x_1,\ x_2,\ x_3$ відповідно, тоді після проходження синапсів і дендритів до нейрона надходять імпульси $w_1x_1,\ w_2x_2,\ w_3x_3$. Отриманий сумарний імпульс:

$$w_1x_1 + w_2x_2 + w_3x_3 \tag{1.1}$$

Сила вихідного імпульсу дорівнює:

$$y = f(x) = f(w_1 x_1 + w_2 x_2 + w_3 x_3) \quad (2.2)$$

Таким чином, нейрон цілком описується своїми вагами і передатною функцією $f(x)$. Одержавши набір чисел (вектор) як входи, нейрон видає деяке число *y* на виході.

Для визначення залежності сигналу на виході нейрона від зваженої суми сигналів на входах використовують передатну функція $f(x)$. Існують різні передатні функції.

Наприклад порогова передатна функція являє собою перепад і зображена на рисунку 1.2. Якщо вхідне значення менше порогового, то значення функції активації дорівнює мінімальному допустимому, інакше – максимально допустимому.[12]

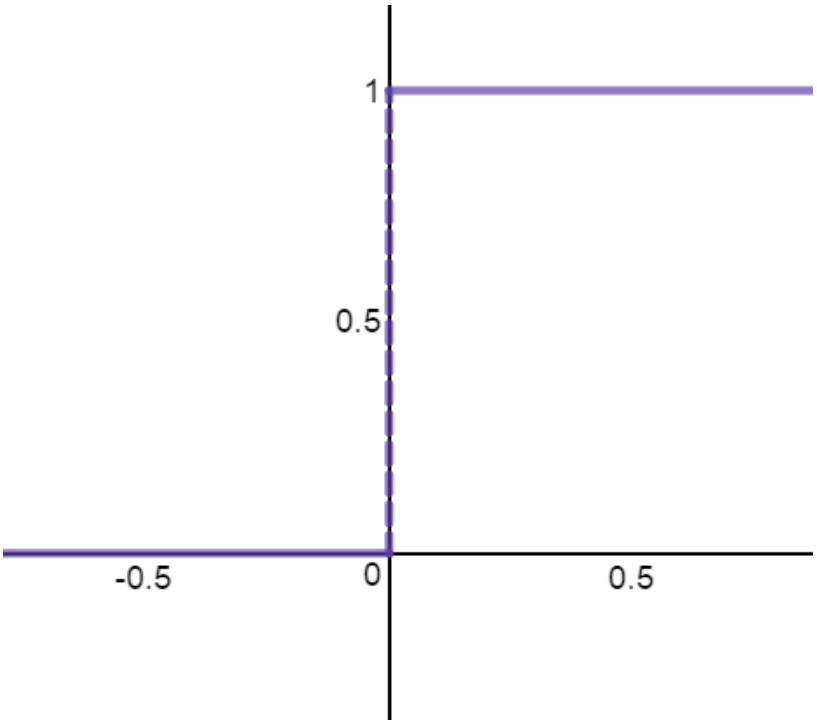

Рисунок 1.2 – порогова функція активації[33]

На рисунку 1.3 зображена лінійна передатна функція. Має дві лінійні ділянки, де функція активації тотожно дорівнює мінімально допустимому і максимально допустимому значенню і є ділянка, на якому функція строго монотонно зростає.[12]

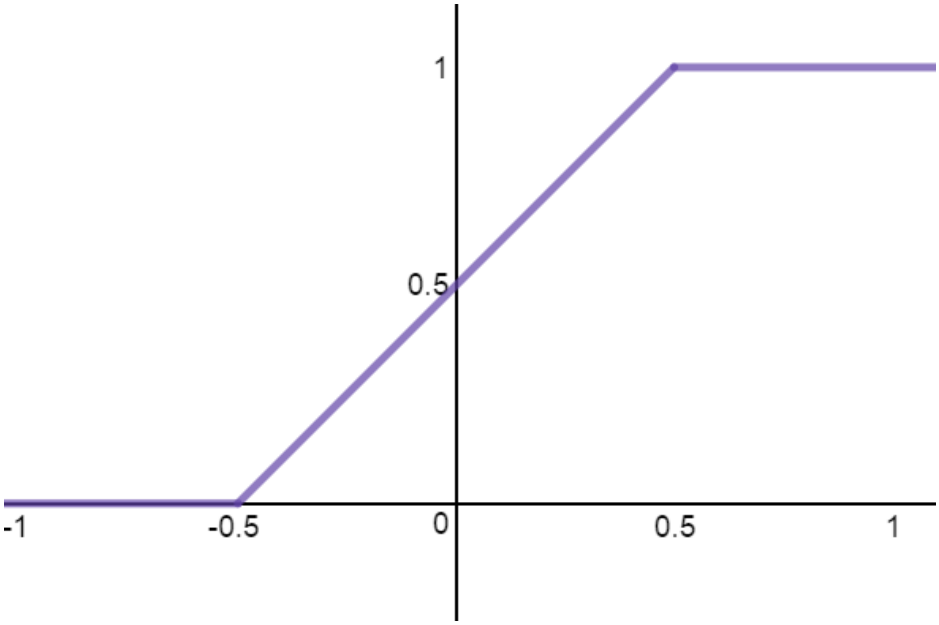

Рисунок 1.3 – лінійна функція активації[33]

На початковій фазі моделювання біологічних нейронних мереж застосовувалися граничні функції (1.2) і (1.3). В даний час найчастіше використовується сигмоїдальна функція, яка зображена на рисунку 1.4 процес активації відбувається плавно, що дає змогу перейти від бінарних виходів до аналогових. Сигмоїд дозволяє підсилювати слабкі сигнали та не насичуватись від сильних сигналів.

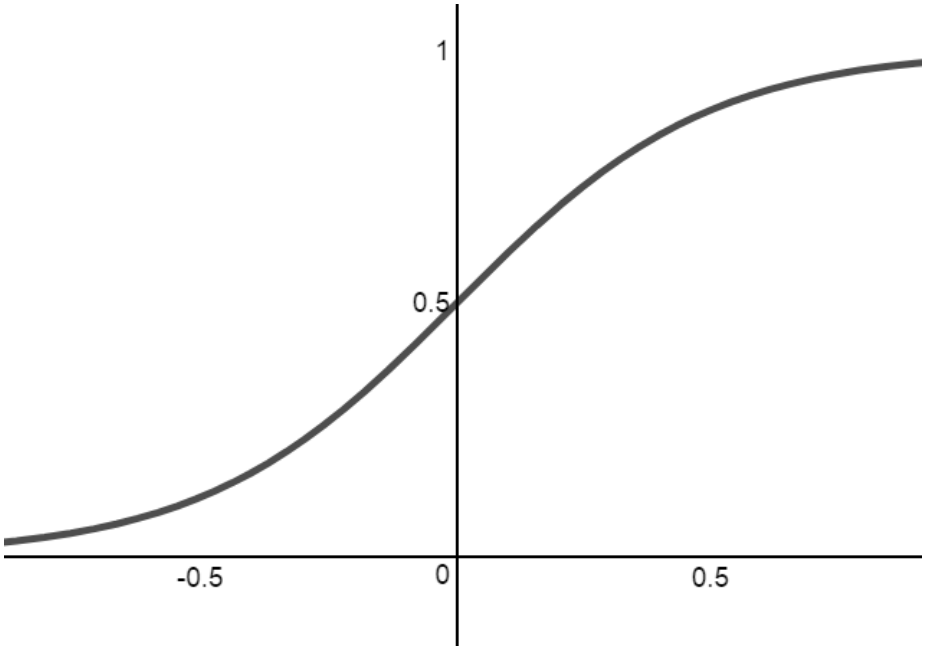

Рисунок 1.4 – сигмоїдальна функція активації[33]

Штучна нейрона мережа (ШНМ) – це математична модель, а також пристрій паралельних обчислень, що представляють собою систему з'єднаних і взаємодіючих між собою простих процесів(штучних нейронів).

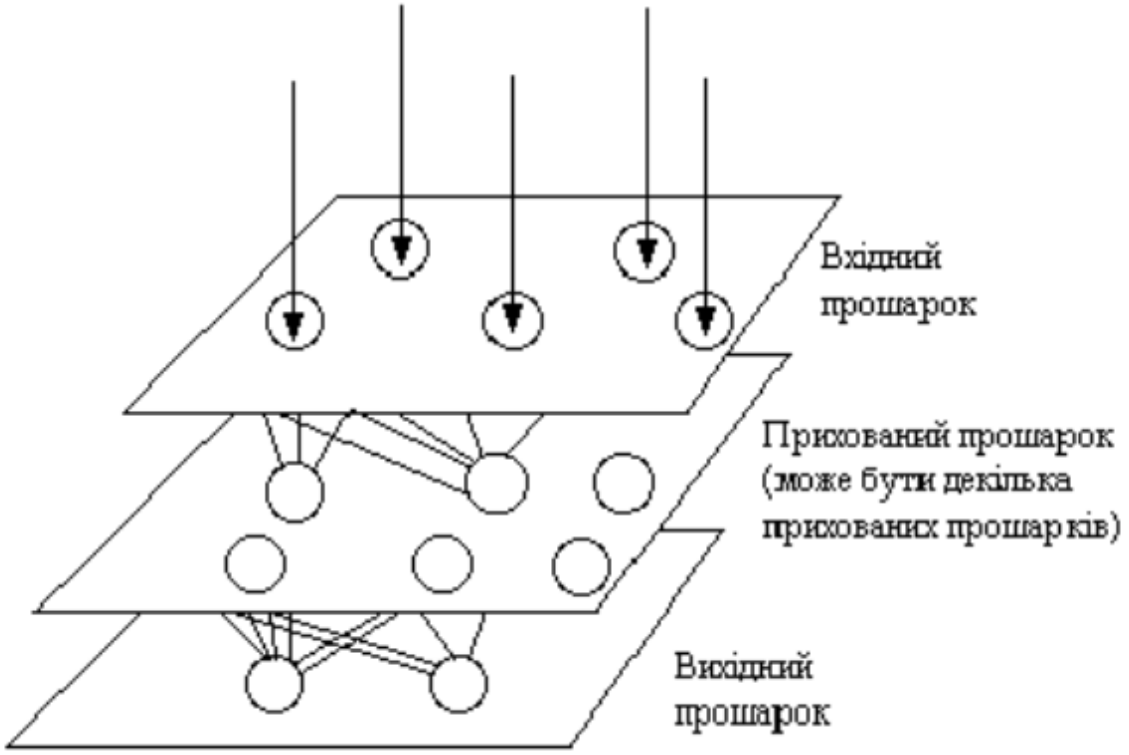

Рисунок 1.5 – схема нейронної мережі[8]

1.2 Генетичні алгоритми

Еволюційна теорія стверджує, що кожен біологічний вид цілеспрямовано розвивається і змінюється для того, щоб краще пристосуватися до навколишнього середовища.

У процесі еволюції багато видів комах і риб придбали захисне фарбування, їжак став невразливим завдяки голкам, людина стала власником дуже складної нервової системи. Можна сказати, що еволюція – це процес оптимізації всіх живих організмів.[8]

Розглянемо, якими ж засобами природа вирішує цю задачу оптимізації. Основний механізм еволюції – це природний відбір. Його суть полягає в тому, що більш пристосовані особи мають більше можливостей для виживання і розмноження і, отже, приносять більше нащадків, ніж погано пристосовані особи.

Завдяки передачі генетичної інформації (генетичному спадкуванню) нащадки успадковують від батьків основні їхні якості. Таким чином, нащадки сильних індивідуумів також будуть відносно добре пристосованими, а їхня частка в загальній масі буде зростати. Щоб було краще зрозуміло принцип роботи генетичних алгоритмів, потрібно зрозуміти як влаштовано механізм генетичного успадкування у природі.[28]

У кожній клітці будь-якої тварини утримується вся генетична інформація цієї особи. Ця інформація записана у виді набору дуже довгих молекул ДНК. Кожна молекула ДНК – це ланцюжок, що складається з молекул нуклеотидів чотирьох типів, що позначаються A, T, C і G.[16]

Власне, інформацію несе порядок проходження нуклеотидів у ДНК. У тваринній клітці кожна молекула ДНК оточена оболонкою – таке утворення називається хромосомою. Кожна вроджена якість особи (колір ока, спадкоємні хвороби, тип волосся і т.д.) кодується визначеною частиною хромосоми, що називається геном цієї властивості. Наприклад, ген кольору ока містить

інформацію, що кодує визначений колір очей. Різні значення гена називаються його аллелями.[8]

При розмноженні тварин відбувається злиття двох батьківських полових кліток і їхні ДНК взаємодіють, утворюючи ДНК нащадка. Основний спосіб взаємодії – кросовер (cross-over, схрещування). При кросовері ДНК предків розділяються на дві частини, а потім обмінюються своїми половинками.

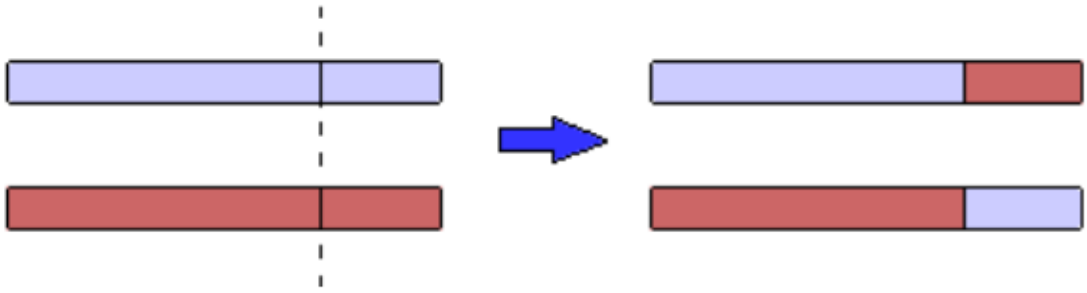

Рисунок 1.6 – схема схрещування[23]

При спадкуванні можливі мутації через радіоактивність або інші впливи, у результаті яких можуть змінитися деякі гени в полових клітинах одного з батьків. Змінені гени передаються нащадку і додають йому нові властивості. Якщо ці нові властивості корисні, вони, швидше за все, збережуться в даному виді.[23]

Ключову роль в еволюційній теорії грає природний відбір. Його суть полягає в тому, що найбільш пристосовані особи краще виживають і приносять більше нащадків, чим менш пристосовані.

Генетичний алгоритм – це проста модель еволюції в природі, яка реалізована у виді комп'ютерної програми. У ньому використовуються як аналог механізму генетичного спадкування, так і аналог природного відбору. При цьому зберігається біологічна термінологія в спрощеному виді:

- хромосома – вектор з нулів і одиниць. Кожна позиція(біт) називається геном;
- генетичний код – набір хромосом є варіантом рішення задачі;
- кросовер – операція обміну хромосомами своїх частин;

- мутація – випадкова зміна однієї або декількох позицій в хромосомі.

Генетичний алгоритм імітує еволюцію популяції як циклічний процес схрещування індивідуумів і зміни поколінь. Життєвий цикл популяції – це кілька випадкових схрещувань (за допомогою кроссовера) і мутацій, у результаті яких до популяції додається якась кількість нових індивідів. Відбір у генетичному алгоритмі – це процес формування нової популяції зі старої, після чого стара популяція гине. Після відбору до нової популяції знову застосовуються операції кросовера і мутації, потім знову відбувається відбір, і так далі.[29]

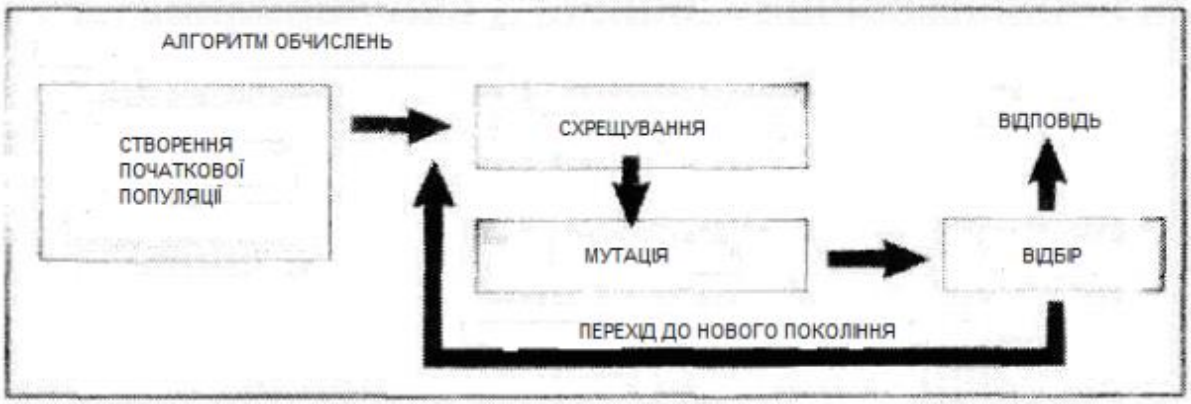

Рисунок 1.7 – структурна схема роботи генетичного алгоритму[29]

1.3 Програмована логічна інтегральна схема – як елементна база для оптимізації вагових коефіцієнтів нейромережі

Програмована логічна інтегральна схема – це електронний компонент, який використовується для створення інтегральних цифрових схем. На відміну від логічного елемента, який має фіксовану функцію, ПЛІС має не визначену функцію під час виготовлення. Перед тим як ПЛІС може бути використана в схемі вона повинна бути запрограмована.[3]

Основні типи ПЛІС:

- FPGA (Field-Programmable Gate Array) – Програмована користувачем вентильна матриця, ПКВМ;

- CPLD (A complex programmable logic device) – Комплекс програмованого логічного пристрою;
- PAL (Programmable Array Logic) – Програмовані матриці логіки.

ПЛІС має широке використання для побудови різних за складністю і можливостями цифрових пристроїв[4], наприклад:

- пристроїв з великою кількістю портів введення-виведення;
- пристроїв, призначених для проектування інтегральних схем спеціального призначення;
- пристроїв, що виконують передачу даних на високій швидкості;
- пристроїв, що виконують цифрову обробку сигналу;
- пристроїв, що виконують криптографічні операції, систем захисту інформації;
- пристроїв, що виконують моделювання квантових обчислень;
- пристроїв, що виконують роль мостів (комутаторів) між системами з різною логікою і напругою живлення;
- реалізацій нейрочипів.

Альтернативою ПЛІС є:

- БМК (Uncommitted Logic Array) – базові матричні кристали, що вимагають виробничого процесу на заводі для програмування. За своєю структурою вони є великими інтегральними схемами, які програмуються технологічно шляхом нанесення маски з'єднань останнього шару металізації;
- ASIC (application-specific integrated circuit) – спеціалізовані замовні інтегральні схеми для вирішення конкретного типу задач;
- спеціалізовані процесори або мікроконтролери (повільніші аніж ПЛІС).

При реалізації нейрообчислювачів сьогодні, як правило використовується гібридна схема, коли блок матричних обчислень реалізується на базі кластерного з'єднання процесорів цифрових сигналів, а логіка управління на основі ПЛІС. Далі

матричне ядро буде реалізовуватися на базі нейрочипів, а сигнальні процесори і ПЛІС залишаться основою для побудови логіки управління.[5]

В даний час безліч фірм в світі займається розробкою і випуском різних ПЛІС, проте, лідирують дві компанії Xilinx та ALTERA. Особливо виділити продукцію будь-якої з цих фірм неможливо, адже за технічними характеристиками вони дуже мало чим відрізняються.

Для прикладу на рисунку 1.8 зображено зовнішній вигляд плати Kintex-7 приблизно на 400 комірок фірми Xilinx.

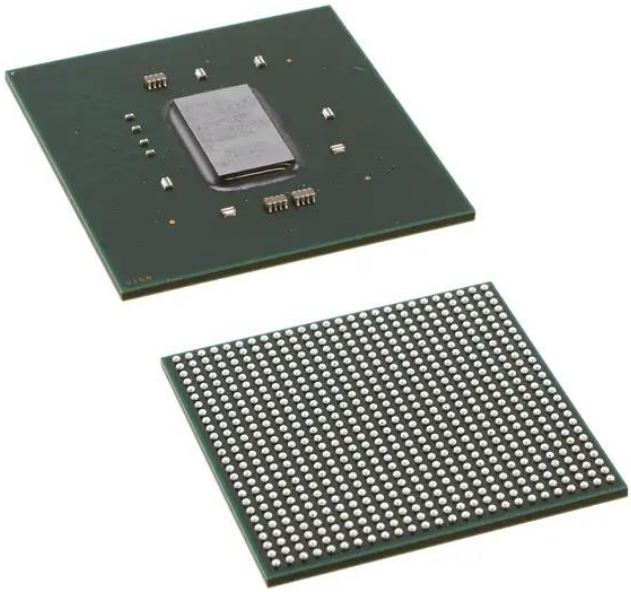

Рисунок 1.8 – Xilinx Inc. Kintex-7 XC7K160T-2FFG676C FPGA Mini Development Learn Core Board[6]

Зовнішній вигляд ПЛІС фірми ALTERA схожий на зовнішній вигляд фірми Xilinx і має такі основні особливості:

- значний обсяг ресурсів – до 170 тис. системних вентилів на кристал;
- висока продуктивність з системними частотами;
- короткий цикл проектування і швидкий час компіляції;
- конкурентоспроможна вартість;

- можливість програмування безпосередньо в системі;
- розвинені і недорогі засоби проектування;
- широка номенклатура кристалів за типом виконання;
- низьке енергоспоживання;
- можливість перекладу проектів в замовні схеми ALTERA.

При виготовленні ПЛІС фірмою Xilinx використовуються три основні технології:

- на основі SRAM (тип FPGA), при цьому конфігурація ПЛІС зберігається у внутрішньому "тіньовому" ОЗУ, а ініціалізація через зовнішні масиви пам'яті. За даною технологією виконані серії: Spartan, Virtex, XC3200, XC4200, XC5200;
- на основі FLASH (тип CPLD), в даному випадку конфігурація зберігається у внутрішній незалежній FLASH – пам'яті і в будь-який момент часу може бути перевантажена безпосередньо з PC. За даною технологією виконана серія XC9500;
- на основі EEPROM (тип CPLD), в даному випадку конфігурація зберігається у внутрішній незалежній EEPROM – пам'яті і в будь-який момент часу може бути перевантажена безпосередньо з ПЕОМ. За даною технологією виконана серія CoolRunner.

Розглянувши та проаналізувавши предметну область можна винести певні результати.

Отже, штучний нейрон та штучна нейронна мережа не що інше як математична модель, що базується на використанні законів математичного аналізу та лінійної алгебри. Генетичні алгоритми створені на основі еволюційної теорії, що стверджує — виживає сильніший. Використовуючи при цьому різні прийоми відбору найсильніших осіб: схрещування, мутація, селекція. Для того, щоб апартно реалізувати потрібний нам алгоритм потрібно використати ПЛІС за

допомогою якої ми отримаємо надзвичайно високі показники швидкодії порівняно з програмною реалізацією завдяки паралельним обчисленням.

Також певну роль грає сама схема, адже від її характеристик напряму залежить швидкість знаходження рішення використовуючи той чи інший алгоритм.

2 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

2.1 Використання генетичних алгоритмів

Генетичні алгоритми в різних формах використовуються для вирішення багатьох наукових задач і технічних проблем. Генетичні алгоритми застосовуються при створенні інших обчислювальних структур, наприклад, автоматів або мереж сортування. В машинному навчанні вони використовуються при проектуванні нейронних мереж або керування роботами. Вони також застосовуються при моделюванні розвитку в різних предметних областях, включаючи біологічні(екологія, імунологія і популярна генетика) і соціальні(економіка і політичні системи) системи.[29]

Тим не менш, можливе популярне використання генетичних алгоритмів. Більшість реальних задач можуть бути сформульовані як пошук оптимального значення, де значення – складна функція, яка залежить від певних вхідних параметрів. В деяких випадках, потрібно знайти ті значення параметрів, при яких досягається точне значення функції. В інших випадках, точний оптимум не потрібен – рішенням може рахуватися будь-яке значення за певну задану величину. В цьому випадку, генетичні алгоритми – прийнятний метод для пошуку “прийнятних” значень.

Сила генетичного алгоритму полягає в його можливості маніпулювати одночасно багатьма параметрами, яка використовується в сотнях прикладних програм, включаючи проектування літаків, налаштування параметрів алгоритмів і пошуку стійких станів систем нелінійних диференційних рівнянь.

2.2 Стратегії навчання нейронних мереж

Під навчанням штучної нейронної мережі розуміється процес налагодження вагових коефіцієнтів $w_{i,j}$ її базових елементів, результатом якого є виконання нейронною мережею конкретних задач – розпізнавання, оптимізації, апроксимації, управління. Досягнення подібних цілей формалізується критерієм якості Q, мінімальне значення якого відповідає якнайкращому вирішенню поставленої задачі.[13]

Різноманіття алгоритмів навчання визначається функціональним призначенням мережі, її архітектурою і вибраною стратегією навчання.

Розрізняють три основні стратегії навчання: «з вчителем», самонавчання і змішану.

У першому випадку нейромережа налагоджується за заданою навчальною вибіркою, відповідно до прийнятого правила або алгоритму.

У другому випадку наперед не вимагається знати правильний результат навчання і в процесі налагодження вагових коефіцієнтів утворюється внутрішня структура активованих базових елементів, що відповідає пред'явленому вектору входу мережі.

При змішаній стратегії навчання частина вагових коефіцієнтів $w_{i,j}$ налагоджується по заданій навчальній вибірці, а інша – відповідно до правил самонавчання.

Перерахованим стратегіям відповідають конкретні алгоритми навчання багаторівневих нейронних мереж(далі БНМ). Кожен алгоритм орієнтований на вирішення певного типу задач та конкретну архітектуру мережі.

Так, БНМ для вирішення задач класифікації, апроксимації і управління навчаються за першою стратегією. Самонавчання використовується в мережах Хопфілда, мережах Кохонена, а змішана стратегія навчання застосовується в RBF-мережах.[13]

2.3 Методи оптимізації вагових коефіцієнтів нейромереж

Методи, що використовуються при навчанні нейронних мереж, багато в чому схожі до методів визначення екстремуму функції кількох змінних. В свою чергу, останні діляться на 3 категорії – методи нульового, першого і другого порядку.[13]

У методах нульового порядку для знаходження екстремуму використовується тільки інформація про значення функції в заданих точках.

У методах першого порядку використовується градієнт функціоналу помилки за налаштованими параметрами:

$$x_{\mathrm{k}+1} = x_{\mathrm{k}} - \alpha_k g_k, \tag{3.1}$$

де x_{k} – вектор параметрів; α_k – параметр швидкості навчання; g_k – градієнт функціоналу, відповідні ітерації з номером k.

Вектор в напрямку, протилежному градієнту, вказує напрямок найкоротшого спуску по поверхні функціоналу помилки. Якщо реалізується рух у цьому напрямку, то помилка буде зменшуватися. Послідовність таких кроків, зрештою, призведе до значень параметрів, що налаштовуються, а це в свою чергу забезпечує мінімум функціоналу.[13]

Певні труднощі тут викликає вибір параметра швидкості навчання α_k. При великому значенні параметра α_k збіжність буде швидкою, але існує небезпека пропустити рішення або піти в неправильному напрямку.

Класичним прикладом є ситуація, коли алгоритм дуже швидко просувається, перестрибуючи потрібне або принаймні прийнятне рішення. Навпаки, при малому кроці, ймовірно, буде обрано вірний напрям, однак при цьому буде потрібно багато ітерацій.

Залежно від прийнятого алгоритму параметр швидкості навчання може бути постійним або змінним. Правильний вибір цього параметра залежить від

конкретного завдання і зазвичай здійснюється дослідницьким шляхом; в разі змінного параметра його значення зменшується у міру наближення до мінімуму функціоналу.

В алгоритмах сполученого градієнта пошук мінімуму виконується вздовж сполучених напрямків, що забезпечує зазвичай більш швидку збіжність, ніж при найшвидшому спуску. Ваги алгоритму сполучених градієнтів на першій ітерації починають рух у напрямку антиградієнта[14]:

$$p_0 = -g_0 \tag{4.2}$$

Потім напрямок наступного руху визначається так, щоб він був пов'язаний з попереднім. Відповідний вираз для нового напрямку руху є комбінацією нового напрямку найшвидшого спуску і попереднього напрямку[14]:

$$p_k = -g_k + \beta_k p_{k-1}, \tag{5.3}$$

де p_k – напрямок руху, g_k – градієнт функціоналу помилки, β_k – коефіцієнт відповідний ітерації з номером k. Коли напрямок спуску визначено, то нове значення вектора параметрів, що настроюються $x_{\mathrm{k+1}}$ обчислюється за формулою:

$$x_{\mathrm{k+1}} = x_{\mathrm{k}} + \alpha_k p_k \tag{6.4}$$

Методи другого порядку вимагають знання других похідних функціоналу помилки. До методів другого порядку належить метод Ньютона. Основний крок методу Ньютона визначається за формулою[14]:

$$x_{\mathrm{k+1}} = x_{\mathrm{k}} - H_k^{-1} g_k \tag{7.5}$$

де x_k – вектор значень параметрів на k-ій ітерації; H – матриця других приватних похідних цільової функції, або матриця Гессе – вектор градієнта на k-ій ітерації.

У багатьох випадках метод Ньютона сходиться швидше, ніж методи спряженого градієнта, але вимагає великих витрат через обчислення матриці Гессе.

Для того щоб уникнути обчислення матриці Гессе, пропонуються різні способи її заміни наближеними виразами, що породжує так звані квазіньютонові алгоритми (алгоритм методу січних площин OSS, алгоритм LM Левенберга - Марквардта).[12]

В багатьох випадках застосування градієнтних методів локального пошуку, традиційно використовуваних для побудови нейромоделей, у ряді випадків (багатоекстремальність цільової функції, недиференційованість функцій активації нейроелементів та ін.) є неприйнятним або неможливим.

Тому доцільним є використання методів еволюційного пошуку, які не вимагають обчислення значень похідних цільових функцій, що дозволяє ефективно застосовувати їх для вирішення задач побудови нейромоделей.

У наш час методи еволюційної оптимізації використовуються для вирішення різних завдань, пов'язаних із синтезом нейромереж: відбір ознак, налаштування вагів, вибір оптимальної архітектури мережі, адаптація навчального правила, ініціалізація значень вагових коефіцієнтів, витяг правил з побудованої мережі, пошук оптимальних значень параметрів досліджуваної системи за наявною нейромоделлю.

Для вирішення задачі управління складними динамічними об'єктами за допомогою нейромережевих регуляторів та моделей потрібно щоб нейромережа навчалася «в темпі» з динамікою об'єкта управління, тобто потрібні алгоритми які за найменший відрізок часу навчать мережу та можуть бути легко розпаралелені.[12]

Проаналізувавши низку методів та стратегій можна винести певні результати.

Отже, генетичний алгоритм можна широко використовувати в повсякденному житті для вирішення різноманітного роду задач екології, генетики, економіки. Існують різні стратегії навчання нейронних мереж: «з вчителем», самонавчання та змішана.

Використовуючи ці стратегії можна навести різні приклади методів які оптимізують вагові коефіцієнти нейронної мережі:

- Градієнт функціоналу помилки за налаштованими параметрами;
- Сполучений градієнт;
- Метод Ньютона;
- Методи еволюційного пошуку.

3 МЕТОД АПАРАТНОЇ РЕАЛІЗАЦІЇ ГЕНЕТИЧНОГО АЛГОРИТМУ

3.1 Апаратна частина для реалізації генетичного алгоритму

Методи та технології які використовуються для розробки застосунків з використанням програмованих логічних інтегральних схемах основані на описі алгоритму спеціальною мовою, яка називається мовою опису апаратури, сьогодні серед лідерів знаходяться VHDL та Verilog, вони виконують автоматичну трансляцію цього опису в опис на рівні логічних таблиць, та інших функціональних компонентів ПЛІС.

Як правило, елементарні функції в ПЛІС реалізуються як окремі проекти чи замовлені віртуальні модулі, в яких вказується розрядність даних та іноді – внутрішня будова.

Алгоритми та методи апаратно-програмної реалізації компонентів нейронних систем управління мають бути орієнтовані на реалізацію в ПЛІС, вони повинні бути адаптовані до їх елементного базису.[21]

ПЛІС різних виробників мають в своєму складі велику кількість різних компонентів таких як: логічні таблиці, тригери, елементи суматорів, блоки оперативного запам’ятовуючого пристрою та постійного запам’ятовуючого пристрою. Ці елементи є основою для реалізації всіх інших пристроїв.[21]

Для конкретизації методів, алгоритмів та критеріїв їх оптимізації була вибрана архітектура ПЛІС фірми Xilinx. Параметри критеріїв, такі як апаратні витрати та витрати часу, можуть бути перераховані для інших виробників.

Серед сучасних розробок, виконаних на ПЛІС високої інтеграції, можна відзначити, перш за все, «нейрочіп-2000», на базі ПЛІС Virtex / Virtex-E (рисунок 3.1).

У розробці цих двох партнерів знаходиться і ряд інших виробів: перспективна розробка «нейрочіп-8» інструментальну плату XDSP-680, на базі ПЛІС сімейства Spartan компанії Xilinx з нейромережевою прошивкою, а також

цілий набір інструментальних плат і на базі ПЛІС різних серій і модулів різного призначення, що дозволяють швидко і ефективно створювати обчислювальні системи різного функціонального призначення.

Всі зазначені розробки засновані на нейронах з пороговою функцією активації і призначені для обчислювальних систем загального призначення. На даний час інтерес представляє розробка нейроконтролерів здатних функціонувати і адаптуватися в реальному часі.

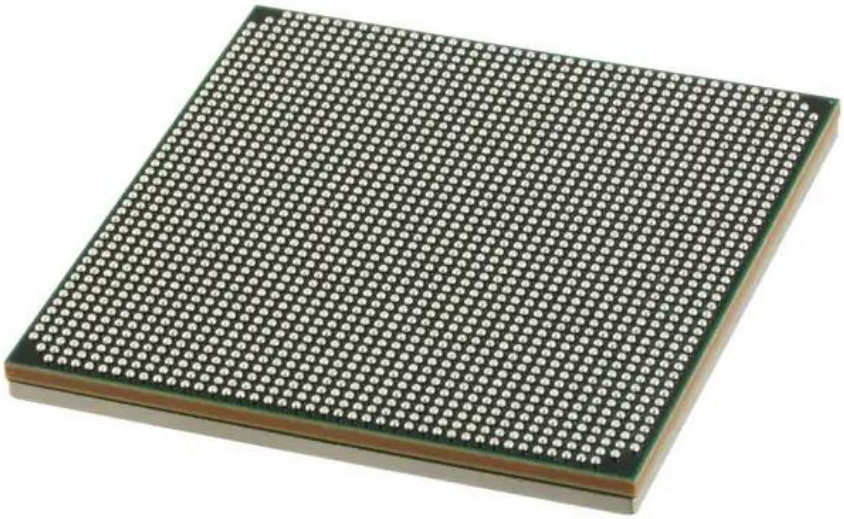

Рисунок 3.1 – інструментальна плата XC7V2000T на базі Virtex-7[7]

Однією з багатьох проблем при апаратно-програмній реалізації нейромережевих систем управління є реалізація алгоритму навчання нейронної мережі та його функцій засобами програмованої користувачем вентильної матриці.

3.2 Навчання нейронних мереж за допомогою генетичного алгоритму

Розглянемо загальну схему навчання нейронних мереж, яка показана на рисунку 3.2, у відповідністю з якою слід вести навчання:

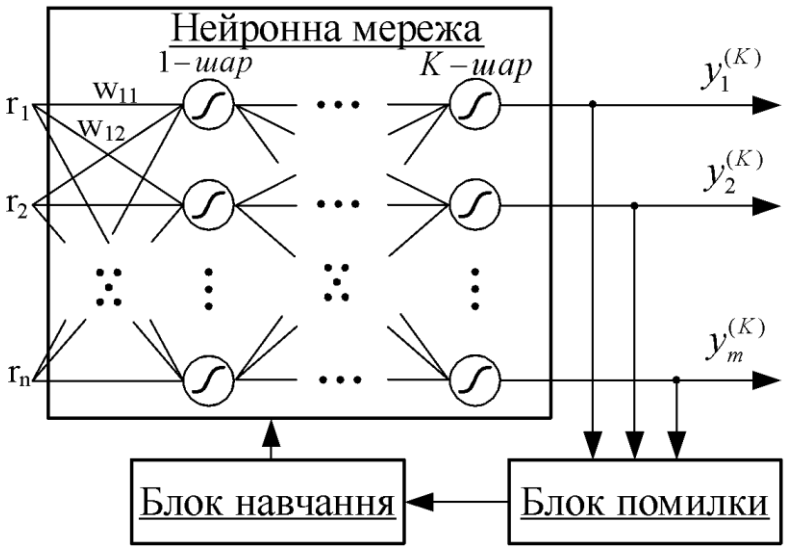

Рисунок 3.2 – схема навчання нейронної мережі[12]

«Блок помилки» порівнює значення виходів мережі, отриманих після подачі на вхід заданих сигналів, і виходів, які повинні вийти при подачі на вхід еталонних сигналів. Значення помилки визначається по формулі.[12]

$$Q = Y_i^{target} - Y_i^{output} \quad (3.1)$$

Одним з найвідоміших на даний момент представником еволюційних алгоритмів є генетичний алгоритм і за своєю суттю є набором інструкцій для знаходження глобального екстремуму багатоекстремальної функції.

Він полягає в паралельній обробці безлічі альтернативних рішень. При цьому пошук концентрується на найбільш перспективних з них. Це говорить про можливість використання генетичних алгоритмів при вирішенні будь-яких завдань штучного інтелекту, оптимізації та прийняття рішень.

Еволюційна оптимізація ваг нейромережі може бути виконана в такій послідовності:[12]

Крок 1. Виконати ініціалізацію початкової популяції хромосомами.

Крок 2. Оцінити хромосоми поточної популяції.

Крок 3. Перевірити критерії закінчення пошуку. У випадку, якщо критерії зупинення задоволено, виконати перехід до кроку 7.

Крок 4. Вибрати особини для генерації нових рішень.

Крок 5. Застосувати оператори схрещування та мутації для хромосом.

Крок 6. Сформувати нове покоління з елітних хромосом. Перейти до виконання кроку 2.

Крок 7. Зупинення.

Розмір хромосоми визначається за формулою:

$$K = N_1(L+1) + \sum_{\mu=1}^{M} N_\mu\,(N_{\mu-1}+1), \tag{3.2}$$

де N_μ – кількість нейронів на μ-ому шарі; L – кількість ознак у навчальній вибірці; M – кількість шарів нейромережі.

Рисунок 3.3 – схематичне подання хромосоми при параметричному синтезі нейромоделей[12]

Для апаратної реалізації еволюційних методів навчання нейронних мереж на FPGA потрібно розпаралелити та пристосувати для FPGA всі потрібні операції.

При розробці паралельних обчислювальних систем необхідно дослідити характеристики алгоритмів які плануємо реалізувати.

Підготовка апаратно-програмної реалізації обрахунків процедури навчання нейронної мережі за еволюційним методом здійснюється на основі графа[12]:

$$\mathrm{GDF} = (A,\ D), \tag{3.3}$$

де A – множина вершин, що відповідає операціям; D – множина дуг, відповідних потокам даних.

Також для апаратно-програмної реалізації еволюційної оптимізації синаптичних ваг нейромережі на FPGA необхідно розробити генератор випадкових чисел.

Генерація випадкових чисел проходить на основі мультиплікативного конгруентного методу. Кожне наступне число формується на базі попереднього за формулою[12]:

$$r_{i+1} = \mathrm{mod}(k * r_i,\ M), \tag{3.4}$$

де M – модуль ($0 < M$); k – множник ($0 \leq k < M$).

Для якісного генератора необхідно підібрати коефіцієнти. Необхідно щоб число M, було досить велике, оскільки період генерації чисел не може мати більше ніж M елементів.

З іншої сторони операція ділення, є доволі повільною, а для FPGA не є стандартною і потребує додаткової реалізації, тому логічно обрати $M = 2^N$, оскільки у цьому випадку операція ділення може бути замінена операцією зсуву[12].

Але для генетичного алгоритму, необхідні випадкові числа в діапазоні від 0 до 7. Їх можна отримати за формулою:

$$r_{i+1} = \mathrm{mod}(r_i,\ 8), \tag{3.5}$$

Тобто для генетичного алгоритму використовуємо 3 молодші розряди згенерованого випадкового числа.

Це можливо зробити, оскільки існує теорема, про те, що, при використанні конгруентного метода генерації випадкових чисел, молодші розряди отриманого випадкового числа ведуть себе так само випадково, як і старші.[21]

Число 8, обумовлене кількістю бітів у хромосомі. Якщо змінити кількість біт, то виникне необхідність генерації випадкових чисел у іншому діапазоні.

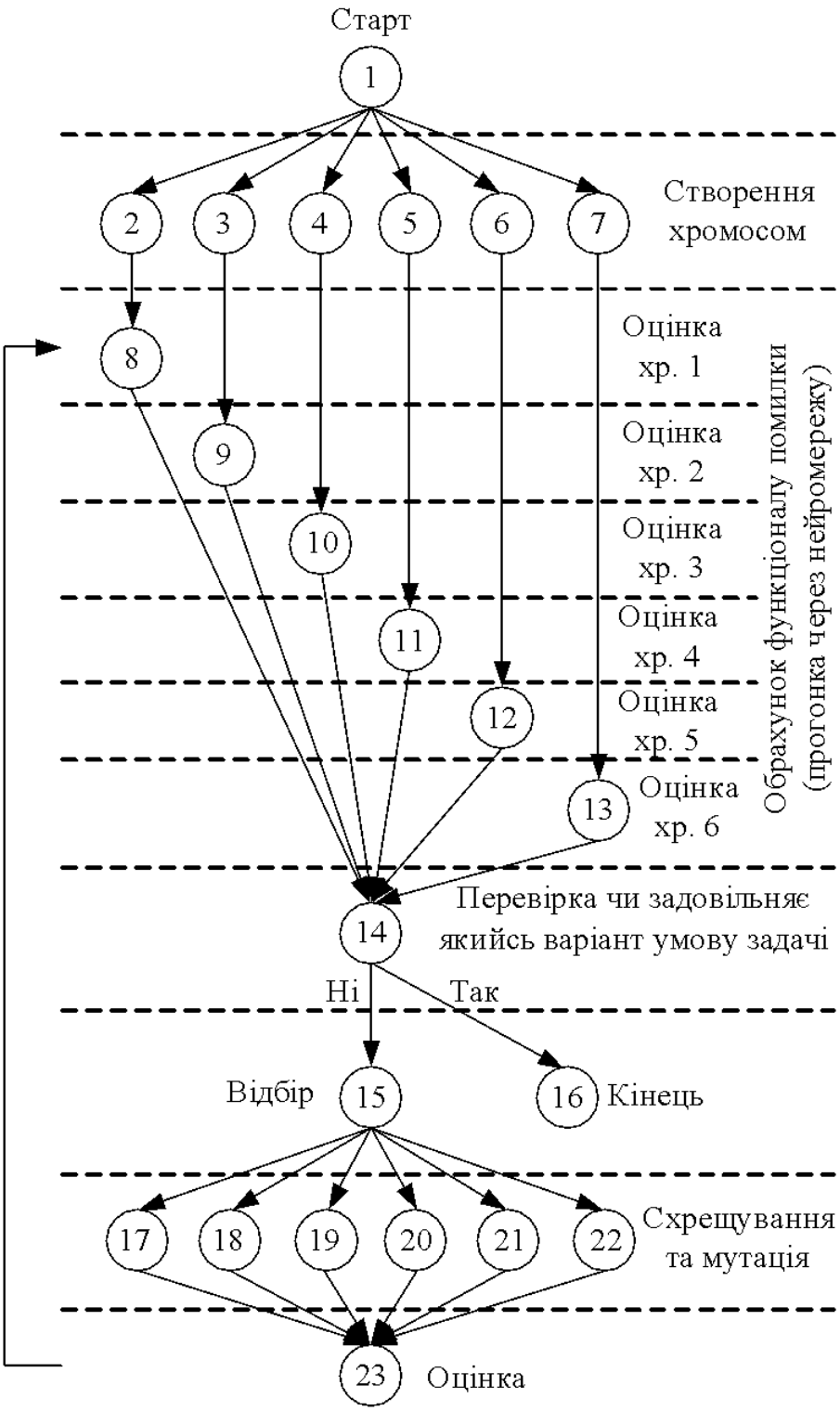

Рисунок 3.4 – граф, що демонструє роботу апаратної версії генетичного алгоритму[13]

Проаналізувавши апаратну та програмну частини можна винести певні результати.

Отже, на швидкодію нейронів та швидкість навчання нейронних мереж, точність результатів навчання, витрачений об’єм пам’яті впливає певна кількість факторів:

– Обчислювальний пристрій;
– Кількість нейронів;
– Кількість шарів;
– Складність обчислень нейронів;
– Алгоритм навчання.

Також певну роль грає сам розмір архітектури, адже він може збільшити час однієї ітерації, але у той же момент зменшити загальний час навчання мережі.

4 СПОСІБ АПАРАТНОЇ РЕАЛІЗАЦІЇ ГЕНЕТИЧНОГО АЛГОРИТМУ

4.1 Опис основних функцій генетичного алгоритму

Основні функції генетичного алгоритму можуть бути виражені за допомогою трьох операторів: оператору відбору(селекції), оператора схрещування та оператора мутації.

Перед кожною ітерацією схрещування та мутації відбувається процес селекції, за допомогою якого допускаються до схрещування та мутації найбільш пристосовані особи.

До цих векторів застосовуються «генетичні оператори» (у більшості випадків «схрещування»- crossover і «мутація»-mutation), створюючи в такий спосіб наступне «покоління». Особи наступного покоління також оцінюються, потім виробляється селекція, застосовуються генетичні оператори і т.д.[8]

Так моделюється «еволюційний процес», що продовжується кілька життєвих циклів (поколінь), поки не буде виконаний критерій зупинки алгоритму.

Таким критерієм може бути:

- отримання глобального, або субоптимального рішення;
- вичерпання числа поколінь, відпущених на еволюцію;
- вичерпання часу, відпущеного на еволюцію.

На першому кроці виконання алгоритму потрібно випадковим чином створити якусь початкову популяцію; навіть якщо вона виявиться зовсім неконкурентоспроможною, генетичний алгоритм всеодно досить швидко переведе її в життєздатну популяцію.

Таким чином, на першому кроці можна особливо не намагатися зробити занадто дуже пристосованих індивідуумів, досить, щоб вони відповідали формату популяції, і на них можна було б розрахувати функцію пристосованості (active/fitness function).

Підсумком першого кроку є популяція *H*, що складається з *N* осіб.

На етапі відбору потрібно з усієї популяції вибрати визначену її частку, що залишиться "у живих" на цьому етапі еволюції.

Є різні способи проводити відбір. Ймовірність виживання особи *n* повинна залежати від значення функції пристосованості *Fitness(n).* Сама частка що вижила *S* є параметром генетичного алгоритму і її просто задають заздалегідь.

За підсумками відбору з *N* осіб популяції *H* повинні залишитися *S* осіб, що увійдуть у підсумкову популяцію *H'*. Інші особи гинуть.

Розмноження в генетичному алгоритмі зазвичай полове – щоб зробити нащадка, потрібно декілька батьків. Розмноження в різних алгоритмах визначається по-різному, зазвичай, залежить від представлення даних. Головна вимога до розмноження – щоб нащадок мав можливість успадкувати риси обох батьків, "змішавши" їх яким-небудь досить розумним способом.

Для того щоб провести операцію розмноження, потрібно вибрати *(1-S)p/2* пар гіпотез з *H* і провести з ними розмноження, одержавши по два нащадка від кожної пари (якщо розмноження визначене так, щоб давати одного нащадка, потрібно вибрати *(1 - S)p* пар), і додати цих нащадків у *H'*. У результаті *H'* буде складатися з *N* хромосом.[13]

Особи для розмноження зазвичай обираються з усієї популяції *H*, а не тільки з елементів що вижили на першому кроці. Це робиться через те, що головним недоліком останнього підходу є недолік розмаїтності(diversity) в особах.

Досить швидко виділяється один єдиний генотип, що являє собою локальний максимум, а потім всі елементи популяції програють йому вибір, і вся популяція "забивається" копіями цієї особи.[8]

Є різні способи боротьби з таким небажаним ефектом; один з них – вибір для розмноження не самих пристосованих, але узагалі всіх осіб.

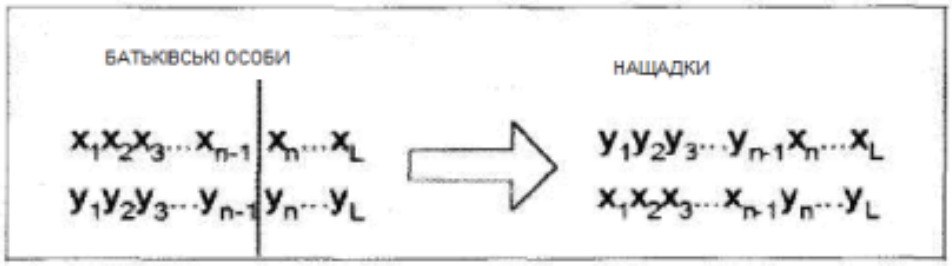

Рисунок 4.1 – обмін (L-n) бітами під час кросинговеру[8]

До мутацій відноситься все те ж саме, що і до розмноження: є деяка частка мутантів *M*, що є параметром генетичного алгоритму, і на кроці мутацій потрібно вибрати *M* осіб, а потім змінити їх відповідно до заздалегідь визначених операцій мутації.

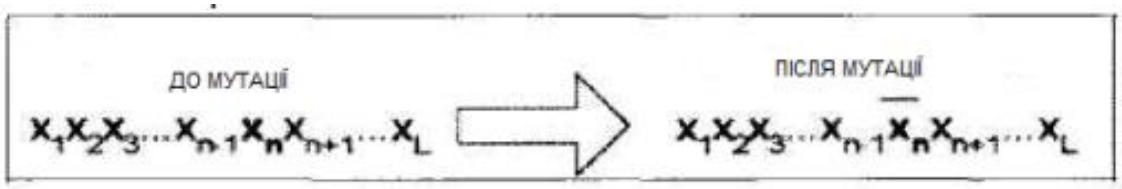

Рисунок 4.2 – зміна n-того біту під час операції мутації[8]

4.2 Апаратна реалізація

Таким чином, можна виділити наступні кроки генетичного алгоритму:

Крок 1. Виконати ініціалізацію початкової популяції хромосомами, що містять інформацію про значення вагових коефіцієнтів мережі заданої структури.

Інформація може бути представлена двовимірним масивом (рисунок 4.3).

Кожен рядок масиву відповідає хромосомі і містить інформацію про весь набір вагових коефіцієнтів мережі. При цьому, кожному нейрону може відповідати різна кількість вагових коефіцієнтів, в залежності від кількості вхідних зв’язків. Таким чином, кожна клітинка масиву відповідає певному ваговому коефіцієнту, що є дійсним числом.

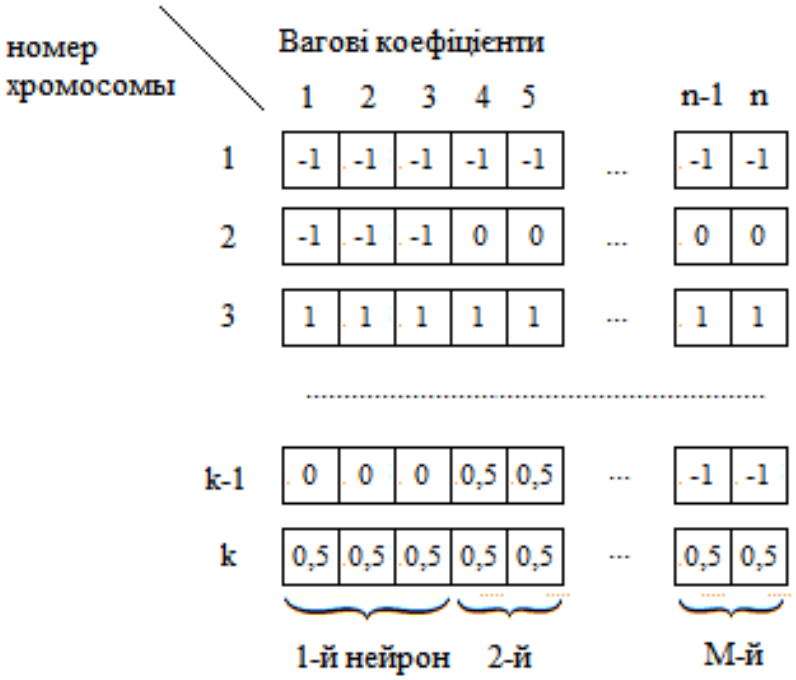

Рисунок 4.3 – представлення інформації про значення вагових коефіцієнтів мережі

Мова VHDL не дозволяє стандартними засобами реалізувати операції з дійсними числами. Дійсні числа було представлено як дріб:

$$\frac{a}{b}, b = 2^c \tag{4.1}$$

Операція ділення на число 2^c, у двійковій арифметиці відповідає зсуву вліво на *c* біт і може бути реалізована за допомогою стандартних засобів VHDL.

Враховуючи попередній абзац, рисунок 4.3 є спрощеним. Замість кожної клітинки на малюнку необхідно зобразити набір бітів, що відповідають числу записаному в клітинці.

Якщо прийняти b=16 і кожен ваговий коефіцієнт представляти восьма бітами, то числу -1 відповідатиме набір бітів «11110000», числу 1 – «00010000», числу 0,5 – «00001000».

Оскільки 11110000b = -16d, 00010000b = 16d, 00001000b = 8d.

І відповідно враховуючи що b=16, -16/b=-16/16=-1, 16/b=16/16=1, 8/b=8/16=0.5.

Таким чином, хромосоми початкової популяції можуть бути представлені у вигляді тривимірного масиву (рисунок 4.4):

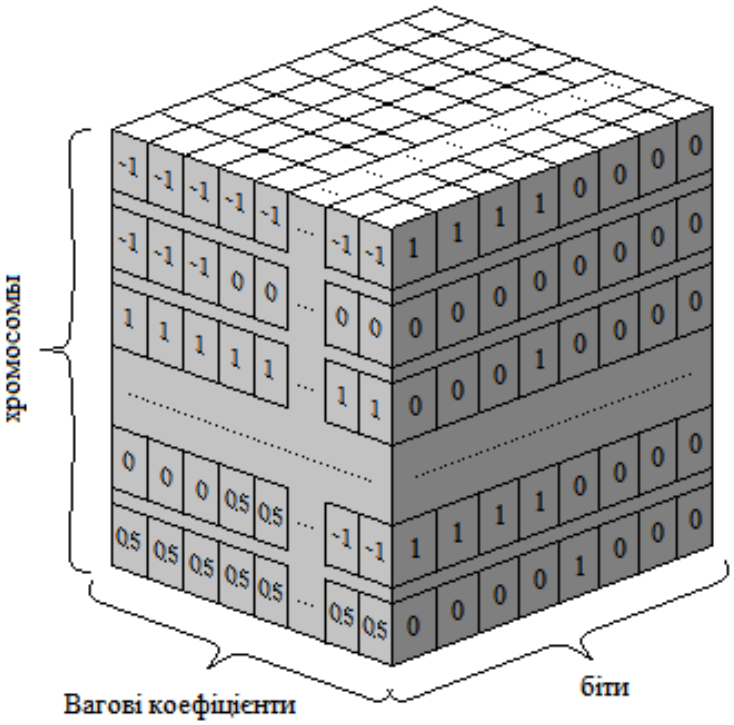

Рисунок 4.4 – представлення вагових коефіцієнтів у вигляді тривимірного масиву

Крок 2. Оцінити хромосоми поточної популяції.

Крок 2.1. Декодувати кожну хромосому популяції в набір вагових коефіцієнтів нейронної мережі. Декодування виконується з використанням стандартних функцій мови VHDL: *To_integer* і *Signed*, таким чином, що:

$$W := \text{To\_integer}(\text{Signed}(\text{chrs}(1)(1))); \quad (4.2)$$

де chrs(1)(1) – перший ваговий коефіцієнт першої хромосоми, масив бітів; W – число типу integer.

Функція *Signed* – перетворює масив бітів у число зі знаком у векторному двійковому представленні. Фактично ця функція не виконує перетворень над бітами числа, вона визначає яке число представлено поданими бітами і як з ним працювати в подальшому. Тип *Signed* задає, що старший біт числа визначає знак, від'ємні числа подаються у додатковому коді, додатні – у прямому.

Функція *To_integer* перетворює масив бітів у ціле число, за алгоритмами визначеними у типі *Signed*.

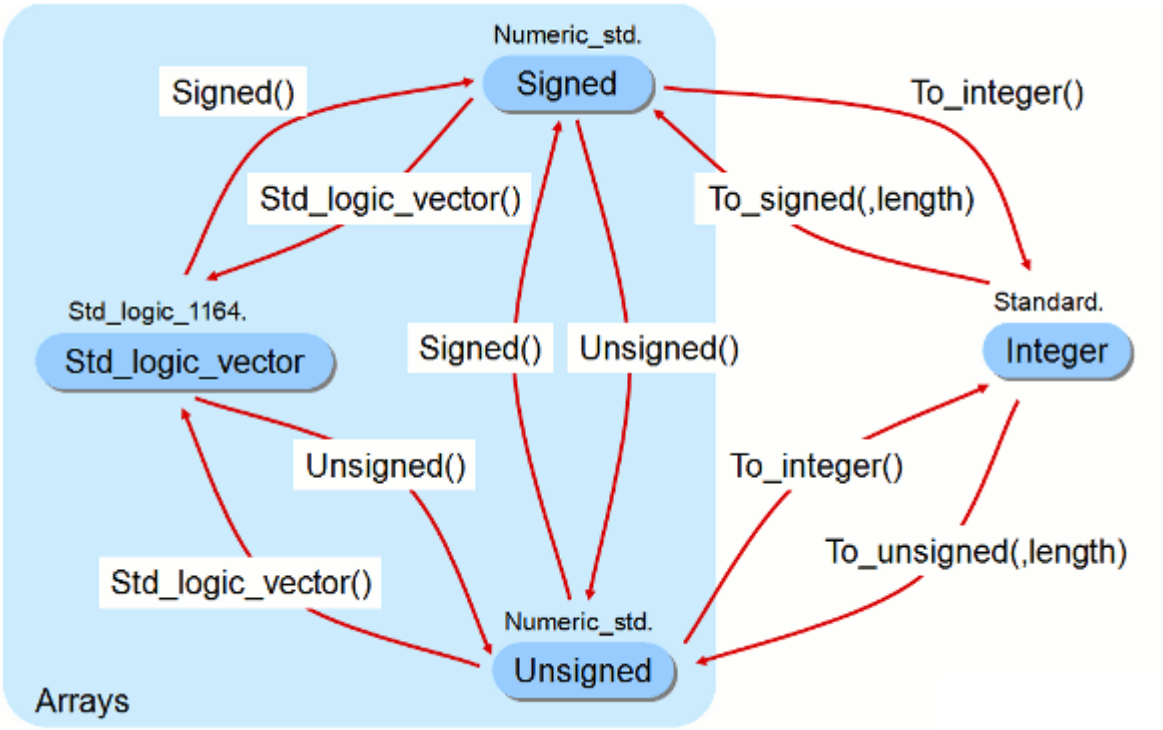

Рисунок 4.5 – стандартні функції перетворення типів мови VHDL

Крок 2.2. Побудувати (прогнати, знайти виходи) нейромережі для розрахованих вагових коефіцієнтів.

Кожному нейрону відповідає окремий процес на мові VHDL, детально розглянуто реалізацію одного нейрона.

Паралельні обчислення, які властиві нейронним мережам, гарно реалізуються на мові VHDL. Кожен процес у мові VHDL має перелік сигналів зміна яких викликає запуск процесу.

На основі цього нейрони (окремі процеси), природнім чином об’єднуються у мережу: виходи попереднього шару мережі запускають процеси, що відповідають наступному шару.

Нейрони першого шару мережі запускають сигнал синхронізації (start), що сповіщає про завершення перетворення хромосоми на вагові коефіцієнти. Зміна сигналів на виході останнього шару мережі запускає процес обчислення фітнес-функцій.

Крок 2.3. Обчислити значення фітнес-функції оцінюваних хромосом, що враховує помилку й складність мережі.

Фітнес-функції визначають наскільки отриманий вихід мережі відрізняється від необхідного, розраховується як різниця між необхідним виходом мережі і отриманим.

Крок 3. Перевірити критерії закінчення пошуку (досягнення прийнятного значення помилки синтезованої нейромережевої моделі, перевищення максимально припустимої кількості ітерацій, перевищення припустимого часу функціонування методу).

У випадку, якщо критерії зупинення виконано, перейти до кроку 7.

Крок 4. Виходячи зі значення фітнес-функції, вибрати особини для генерації нових рішень.

Крок 5. Застосувати оператори схрещування та мутації для хромосом, відібраних на попередньому кроці.

Усі хромосоми розглядаємо як набір бітів. Тоді, перша сума визначає першу хромосому для схрещення (її номер *i*), друга сума – другу (номер *j*):

$$H' = \sum_{i=1}^{n_1} \sum_{j=i+1}^{n_2} \left(\sum_{k=1}^{b-1} H_{ik} 2^k + H_{jb} 2^b + \sum_{k=b+1}^{B} H_{ik} 2^k \right), \tag{4.3}$$

де H' – новий набір хромосом; H – попередній; b і k – номери бітів; 2^b, 2^k – позначають позицію біта у двійковому числі; H_{ik} – k-й біт i-ї хромосоми, H_{jb} – біт з номером b у j-ї хромосомі.

Таким чином, за формулою, усі біти нової хромосоми копіюються з i-ї хромосоми попереднього покоління, окрім біта з номером b, він копіюється з j-ї хромосоми.

Приклад реалізація генетичного алгоритму мовою VHDL:

for i **in** 1 **to** 3 **loop** -- цикл, що відповідає першій сумі у формулі, визначає номер першої хромосоми при схрещенні.

for j **in** i+1 **to** 4 **loop** -- цикл, що відповідає другій сумі у формулі, визначає номер другої хромосоми при схрещенні.

n:=n+1; -- розрахунок номера хромосоми у новій популяції.

for k **in** 1 **to** b-1 **loop**

chrs(n)(k):=chrs2(i)(k); -- chrs - нова популяція хромосом, chrs2 - попередня популяція.

-- копіюються біти з 1 по (b-1)-й i-ї хромосоми попередньої популяції у хромосому нової популяції.

endloop;

chrs(n)(b):=chrs2(j)(b); -- змінна значення якої визначається за межами описаного алгоритму. Копіюється біт з номером b j-ї хромосоми попередньої популяції у хромосому нової популяції.

for k **in** b+1 **to** 24 **loop**

chrs(n)(k):=chrs2(i)(k); -- копіюються біти з (b+1)-го по 24-й i-ї хромосоми попередньої популяції у хромосому нової популяції.

endloop;

endloop;

endloop;

Така реалізація дуже близька до записаної формули, але вона має наступні недоліки:

- Від другої хромосоми наслідується лише один біт, збільшення кількості біт за такого алгоритму – досить важка задача.
- Формула (1) є спрощеною, вона розглядає хромосому, як набір бітів, на кроці 1 було визначено, що хромосома складається з вагових коефіцієнтів, кожен з яких є набором бітів.

```
for i in 1 to 3 loop
 for j in i+1 to 4 loop
   n:=n+1;
   chrs(n):=chrs2(i);
   for m in 1 to 3 loop
    for p in 1 to NumberOfBits/2 loop  -- цикл визначає скільки бітів у новій
```

хромосомі буде наслідовано від другої хромосоми, що приймає участь у схрещенні. NumberOfBits - число бітів у хромосомі. У випадку коли в циклі зазначено NumberOfBits/2 від другої хромосоми наслідується половина бітів.

```
        b = random(NumberOfBits); -- b - випадкове число, що визначає
```

номер біта. Копіюється b-й біт m-го вагового коефіцієнту j-ї хромосоми попередньої популяції у b-й біт m-го вагового коефіцієнту n-ту хромосому нової популяції.

```
          chrs(n)(m)(b):=chrs2(j)(m)(b);
    endloop;
  -- копіювання повторюється NumberOfBits/2 раз, тому від другої
```

хромосоми наслідується NumberOfBits/2 біт.

```
   endloop;
 endloop;
endloop;
```

Формула із урахуванням того, що хромосома складається з вагових коефіцієнтів:

$$H' = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \sum_{m=1}^{m_{length}} \left(\sum_{k=1}^{b-1} H_{ik} 2^k + H_{jb} 2^b + \sum_{k=b+1}^{B} H_{ik} 2^k \right), \quad (4.4)$$

де m – номер вагового коефіцієнту.

Крок 6. Сформувати нове покоління з елітних хромосом і хромосом-нащадків, отриманих шляхом застосування схрещування й мутації. Перейти до виконання кроку 2.

Крок 7. Зупинення.

Хромосома складається з *K* генів, що містять значення ваг і зсувів всіх нейронів мережі. При цьому для подання значень вагових коефіцієнтів у хромосомах застосовується дійсне кодування.

Проаналізувавши реалізацію генетичного алгоритму можна винести певні результати.

Отже, перш за все потрібно закодувати задачу таким чином, щоб її рішення могло бути представлене у виді вектора, після чого створити випадковим чином початкову популяцію, оцінити її та обрати найпристосованіших індивідуумів.

Після чого схрестити та/або мутувати певну кількість хромосом і таким чином створити нове покоління. Повторити виконання з самого початку поки ми не отримаємо потрібне нам рішення задачі. Було докладно описано процес схрещування та мутації.

Також було описано апартно-програмна частина реалізації алгоритму. Покроково розписана реалізація описаних вище процесів та запропоновано декілька варіантів рішення, більш проста, що має наступні недоліки:

- Від другої хромосоми наслідується лише один біт, збільшення кількості біт за такого алгоритму – досить важка задача;

- Вона розглядає хромосому, як набір бітів, на кроці 1 було визначено, що хромосома складається з вагових коефіцієнтів, кожен з яких є набором бітів.

Та більш складна яка враховує, що хромосома складається з вагових коефіцієнтів.

Було надано коротенький код сніпет який демонструє частину, де реалізовано математичну модель згідно з формулою яка демонструє процес навчання нейронної мережі за допомогою генетичного алгоритму.

5 ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ АПАРАТНОЇ РЕАЛІЗАЦІЇ ГЕНЕТИЧНОГО АЛГОРИТМУ

5.1 Блок штучного нейрону на ПЛІС

Розробка блоку генетичного алгоритму складається з декількох частин:

- Блок штучного нейрону який ми будемо навчати;
- Блок генетичного алгоритму.

Таким чином доцільно розказати про пристрій штучного нейрону, а потім описати саму реалізацію інтерфейсу, що поєднує в собі навчання цього нейрону.

Реалізований штучний нейрон з 4-ма входами і сигмоїдальною функцією активації на ПЛІС з використанням 16-розрядних чисел з фіксованою точкою зайняв 672 LUTs (Look Up Table - вентилі логічної матриці).

Штучний нейрон розроблений на ПЛІС як окремий обчислювальній блок зображений на рисунку 5.1.

Швидкодія, як сумарна затримка комбінаційної схеми блоку нейрона, склала 75.6 нс. При зменшенні або збільшенні кількості лінійних відрізків, кількості входів нейрона або зміну розрядності чисел буде змінений і використаний ресурс ПЛІС для одного нейрона, а також точність і швидкодія.

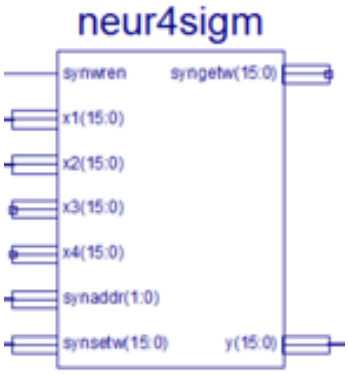

Рисунок 5.1 – блок штучного нейрона в ПЛІС

Кожен нейрон оперує блоком оперативної пам'яті де зберігаються ваги. Для задання ваг у кожному нейроні використовуються такі сигнали:

synaddr (synapse address) – вибір синапсу вага якого буде зчитуватися або записуватися за його номером у двійковому коді. Для нейрона на 4 синапси цей вхід буде 2-х бітним, на 8 - 3-х і т.д.

synsetw (synapse set weight) – значення ваги, яке треба записати в синапс.

synwren (synapse write enabled) – сигнал при наявності 1 на вході якого нейрон записує в обраний синапс значення з synsetw, коли даний сигнал дорівнює 0, нічого не відбувається.

На входи штучного нейрона подаємо значення вхідного вектора, задаємо змінну x, що дорівнює виходу суматора:

$$x = \sum_{i=1}^{N} w_i a_i, \tag{5.1}$$

де a – входи нейрона; w – синаптичні ваги нейрона.

Розрахунок проходить з використанням чисел з фіксованою точкою. Кожне число – 16 біт, з яких 9 приходить на цілу частину і 7 на дробову.

Для арифметичних операцій над цими числами використовується бібліотека fixed_pkg_c.vhdl. Арифметичні операції даної бібліотеки добре оптимізовані.

Розраховуємо модуль від аргументу сигмоїдальної функції, задаємо змінну x'. На даному кроці виконується розбиття на лінійні шматки сигмоїдальної функції активації, і визначаються коефіцієнти лінійних рівнянь k і b. Розбивка на лінійні шматки їх рівняння показана на рисунку 5.2.

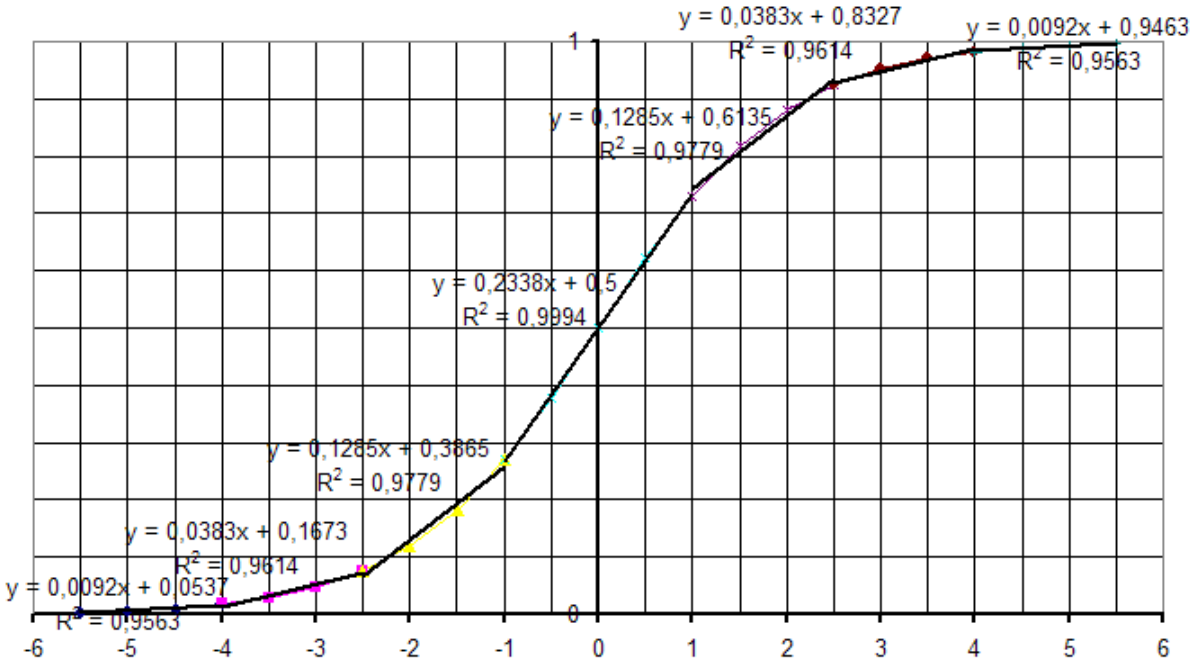

Рисунок 5.2 – кусково-лінійна апроксимація сигмоїдальної функції[30]

Визначаємо коефіцієнти лінійних рівнянь:

$$k, b = [0.234, 0.500], \text{якщо } 0 \leq x' < 1 \quad (5.2)$$

$$k, b = [0.129, 0.605], \text{якщо } 0 \leq x' < 2.5$$

Визначаємо змінну *f*, обчислюємо значення за формулою:

$$f = k * x' + b \quad (5.3)$$

Якщо $x < 0$, обрахувати значення локальної змінної за формулою:

$$f = 1 - f \quad (5.4)$$

Задати значення вихідного сигналу нейрона рівним значенню змінної *f*.

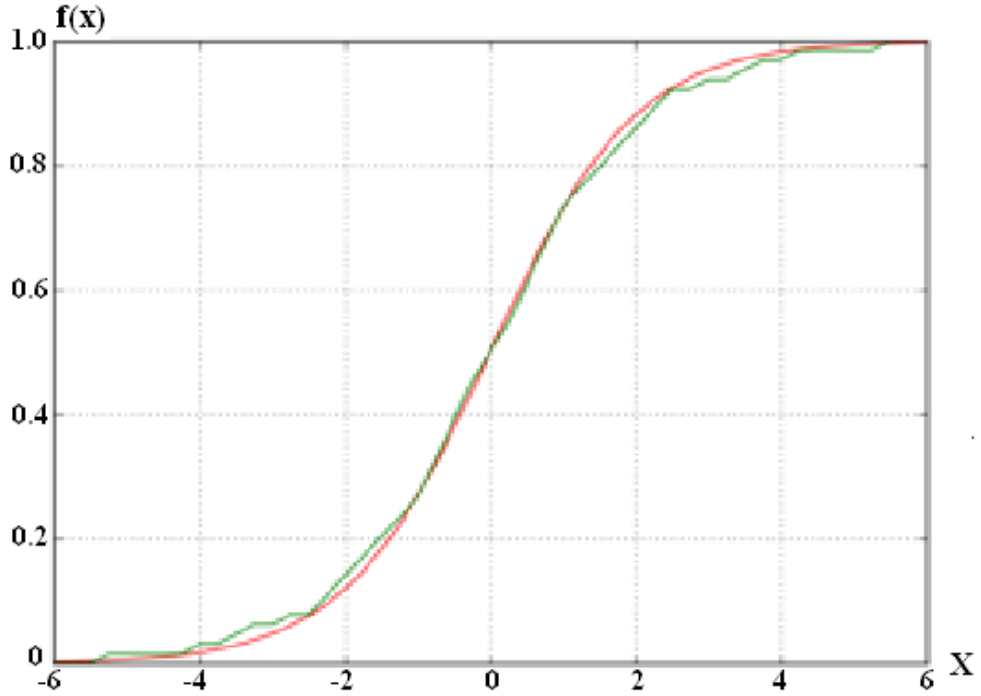

Рисунок 5.3 – сигмоїдальна функція, реалізована в ПЛІС

На рисунку 5.3 представлено класичну сигмоїдальну функцію та функцію реалізовану в ПЛІС за розробленим алгоритмом. Як видно реалізований обчислювальний блок сигмоїдальної функції в ПЛІС досить точно відображає її, для подальшої реалізації штучних нейронних мереж та компонентів нейромережевих систем управління на їх базі.

5.2 Розробка блоку генетичного алгоритму на ПЛІС

Блок генетичного алгоритму складається з портів які приймають сигнали на вхід – in1, in2, in3, порту який зберігає значення яке ми хочемо досягти в процесі навчання – zout. Та порту виходу, що показує потечня значення виходу після кожної ітерації навчання.

FinishTeaching – сигнал, що зберігає інформацію про те, що навчання мережі завершено.

Start – сигнал, що запускає процес, що формує вихід мережі для кожного нового набору вагових коефіцієнтів. Змінює своє значення при запуску цього процесу.

i – номер хромосоми, на основі якої були сформовані вагові коефіцієнти перед запуском процеса, що формує вихід нейронної мережі(перед установкою сигнала start в 1).

NoutsForming – сигнал, який встановлено в 1 поки відбувається навчання хромосом в нейронній мережі. Переходить із 1 в 0 коли всі 6 хромосом оброблено, коли можна переходити до аналізу результатів: завершити алгоритм або за допомогою схрещування та мутації сформувати нове покоління хромосом. Перехід сигнала з 0 в 1 говорить про початок нової ітерації генетичного алгоритму.

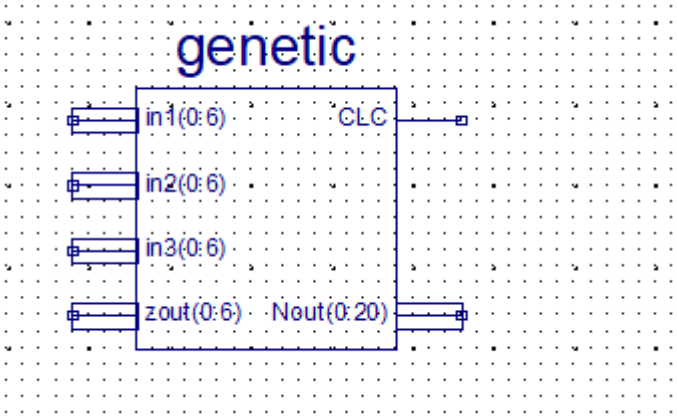

Рисунок 5.4 – блок генетичного алгоритму в ПЛІС

Блок генетичного алгоритму складається з портів які приймають сигнали на вхід – in1, in2, in3, порту який зберігає значення яке ми хочемо досягти в процесі навчання – zout. Та порту виходу, що показує потечня значення виходу після кожної ітерації навчання.

FinishTeaching – сигнал, що зберігає інформацію про те, що навчання мережі завершено.

Start – сигнал, що запускає процес, що формує вихід мережі для кожного нового набору вагових коефіцієнтів. Змінює своє значення при запуску цього процесу.

і – номер хромосоми, на основі якої були сформовані вагові коефіцієнти перед запуском процеса, що формує вихід нейронної мережі(перед установкою сигнала start в 1).

NoutsForming – сигнал, який встановлено в 1 поки відбувається навчання хромосом в нейронній мережі. Переходить із 1 в 0 коли всі 6 хромосом оброблено, коли можна переходити до аналізу результатів: завершити алгоритм або за допомогою схрещування та мутації сформувати нове покоління хромосом. Перехід сигнала з 0 в 1 говорить про початок нової ітерації генетичного алгоритму.

Рішення про перехід до аналізу даних, отриманих після навчання нейронної мережі і рішенням про початок нової ітерації в алгоритмі навчання приймається в двох різних процесах.

Мова VHDL не дозволяє зміну значення одного сигналу в двох різних процесах, тому був введений додатковий сигнал – analysis. Після прогонки 6-ти хромосом через нейронну мережу значення цього сигналу переходить із 0 в 1 а при початку нової ітерації генетичного алгоритму – із 1 в 0. При переході сигналу analysis із 1 в 0, змінюється і NoutsForming, але в іншому процесі.

Analysis – сигнал, що встановлено в 1, поки відбувається аналіз інформації, що була отримана при навчанні хромосом.

Сигнал CLC формує затримки в часі в 1 пікосекунду. Всередині блоку реалізовано оператори схрещування, мутації та селекції наступного покоління хромосом.

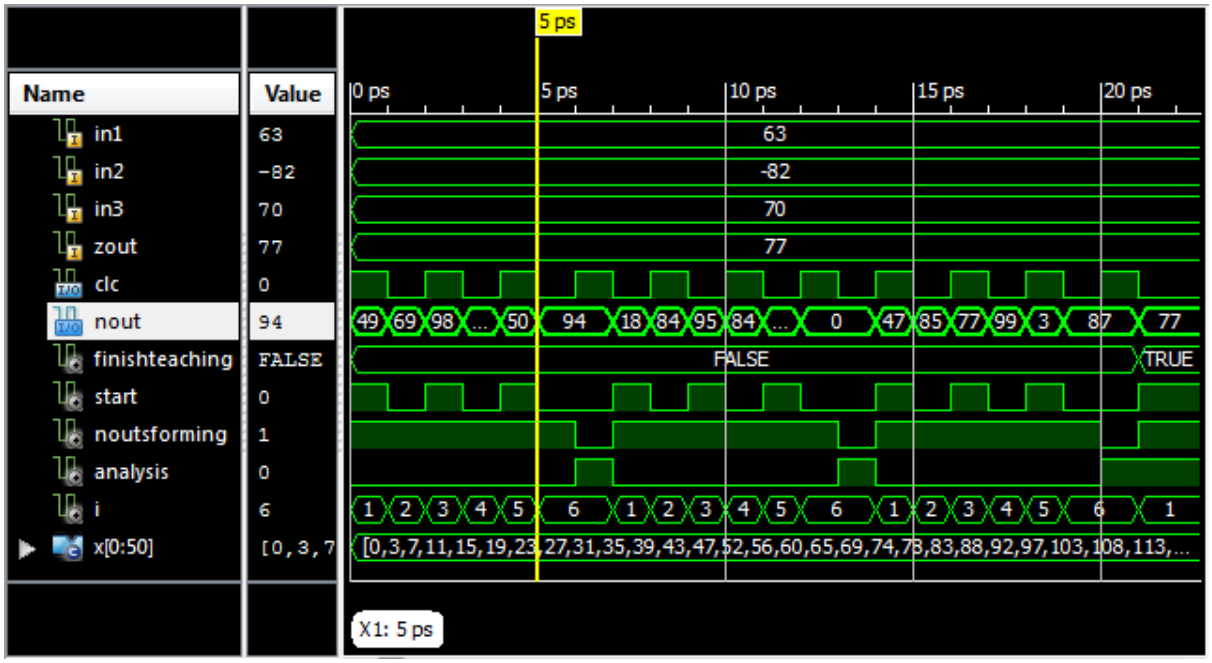

Рисунок 5.5 – результати виконання генетичного алгоритму в ПЛІС

Перші 6 пікосекунд відбувається навчання хромосом нейронної мережі (сигнал NoutsForming=1). В цей час значення сигналу і змінюється від 1 до 6 кожну пікосекунду, на початку кожної пікосекунди змінюється значення сигналу Start і формується вихід нейронної мережі для і-ї хромосоми (змінюється значення сигнала Nout).

На 7-ій пікосекунді відбувається аналіз інформації, отриманій при прогонці хромосом через нейронну мережу(сигнал Analysis=1). Оскільки на цій ітерації не досягнуто необхідне значення виходу (протягом 1-6 пікосекунд Nout не дорівнює Zout), формуються нові хромосоми і запускається нова ітерація генетичного алгоритму.

Протягом 8-14 пікосекунд сигнали і, start, NoutsForming, Analysis, змінюються точно так само як і на попередніх 7-и пікосекунд, проте значення сигналу Nout інше, оскільки алгоритм виконується для іншого покоління хромосом.

На третій ітерації третя хромосома дає необхідне значення виходу мережі, необхідне значення виходу мережі (77), тому після аналізу на 21 пікосекунді сигнал FinishTeaching приймає значення true, алгоритм навчання закінчує своє

виконання, сигнали i, start, NoutsForming, Analysis перестають змінювати свої значення, вагові коефіцієнти мережі приймають відповідні значення, що відповідають третій хромосомі на останній ітерації алгоритму, а вихід мережі встановлюється в необхідне значення 77.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Latches	220	15,360	1%
Number of 4 input LUTs	11,306	15,360	73%
Number of occupied Slices	6,096	7,680	79%
Number of Slices containing only related logic	6,096	6,096	100%
Number of Slices containing unrelated logic	0	6,096	0%
Total Number of 4 input LUTs	11,701	15,360	76%
Number used as logic	11,306		
Number used as a route-thru	395		
Number of bonded IOBs	52	173	30%
Number of MULT18X18s	13	24	54%
Number of BUFGMUXs	2	8	25%
Average Fanout of Non-Clock Nets	2.53		

Рисунок 5.6 – узагальнення щодо використання пристрою

Для простоти були підібрані такі значення in1, in2, in3, zout, що алгоритм завершився за 3 ітерації, в середньому алгоритм навчання з іншими значеннями in1, in2, in3 завершується за 20-70 ітерацій, 200-500 пікосекунд, із затримками, при відсутності затримок, цей алгоритм виконується за 1 пікосекунду.

5.3 Розробка блоку для навчання нейрону з використанням генетичного алгоритму

Так як модуль нейрону треба було навчити за допомогою генетичного алгоритму, було прийнято рішення створити окремий блок neuron_with_trainer який би в собі інкапсулював алгоритм навчання та об’єкт нейрону, що включає в себе функцію активації, а також оцінку критерія скінченності алгоритму.

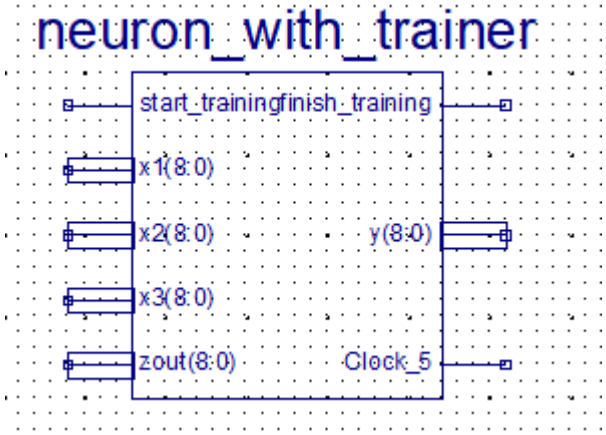

Рисунок 5.7 – блок нейрону, що навчено за допомогою генетичного алгоритму в ПЛІС

Модуль генетичного алгоритму або модуль навчання приймає на вхід числа типу Integer, які в свою чергу можуть бути як додатні так і від'ємні. Різниця між модулем навчання та нейроном можна спостерігати у вхідних значеннях.

Модуль нейрона:

x1, x2, x3, x4: in STD_LOGIC_VECTOR (15 downto 0); (5.5)

де входи x1, x2, x3 та x4 мають довжину в 16 біт, серед яких 7 біт використовуються для представлення дробових чисел. STD_LOGIC_VECTOR означає беззнаковий вектор, а отже не може використовуватися для представлення чисел зі знаком.

Модуль навчання:

in1:in integer range -100 to 100 (5.6)
in2:in integer range -100 to 100
in3:in integer range -100 to 100

де входи in1, in2, in3 представляють знаковий вектор або в загальному масив цілих чисел від -100 до 100.

Модуль генетичного алгоритму виконує оператори схрещування/мутації та ітерує настільки багато наскільки він зможе знайти найкраще рішення для заданого входу хромосом та zout. Zout – це обов'язковий параметр який позначає потрібний вихід нейрона.

Для того, щоб алгоритм почав своє функціонування ми повинні задати значення початку навчання. Використовуючи процес Training_start ми задаємо значення початку навчання, таким чином запускаємо на виконання низку інших процесів.

```
Training_start:process(start_training, FinishTeaching)
  begin
    if(start_training ='1') then
      r_training_start <= '1';
    else if(FinishTeaching ) then
      r_training_start <= '0';
    end if;
end process Training_start;
```

Процес neuron починає виконуватись як тільки значення сигналу start змінить своє значення, а саме після того як процес навчання почнеться. Після цього використовуючи змінну LocalOut ми вказуємо поточний вихід нейрона, який в подальшому буде порівнюватсь з очікуваним результатом змінної zout.

Змінна w_neuron_out позначає поточний вихід мережі, тобто дорівнює y. Після використання деяких перетворень з двійковими числами, ми отримуємо конкретне значення виходу.

```
neuron : process(start) is
  variable LocalOut:integer range 0 to 1023;
begin
  LocalOut := to_integer(unsigned(w_neuron_out(8 downto 0)));
  if (not FinishTeaching) then
    lout(i):=LocalOut;
  end if;
end process neuron;
```

Як тільки навчання починається, ваги починають оновлювати свої значення таким чином, що ми отримуємо необхідне значення виходу.

Процес CntrInI починає своє виконаня коли навчання старує і кожного разу коли ми міняємо значення сигналу і, таким чином міняємо адресу поточного синапсу в який пишемо отримані ваги після кожної ітерації виконання алгоритму.

Це значення ми називаємо лічильником який використовується для synaddr. Таким чином, коли кожного разу значення змінної і змінюється, лічильник також змінює своє значення та записує його в synaddr.

В даному випадку ми перевіряємо чи подія start_training наступила, якщо так, присвоюємо початкове значення адреси, тобто першу адресу. Якщо ж ваги вже ініціалізовано, то ми збільшуємо значення адреси.

```
CntrInI: process (i, start_training, Clock_1)
  begin
    if (start_training'event and start_training = '1') then
      r_counter_in_i <= "00";
    else if (r_weight_initialised = '0' or (Clock_1'event)) then
      r_counter_in_i <= r_counter_in_i + 1;
    else
      r_counter_in_i <= "00";
    end if;
end process CntrInI;
r_synaddr <= r_counter_in_i;
```

Для того щоб ми могли змогу міняти ваги нашої нейронної мережі нам потрібно дати дозвіл на запис значення в синапси. За допомогою змінної r_synwren ми даємо змогу нашому алгоритму зрозуміти, коли саме нам потрібно почати змінювати значення синапсів.

Після того як навчання починається ми даємо дозвіл на запис, а коли навчання завершується ми припиняємо зміни.

```
SynWrEn: process(start_training,FinishTeaching)
  begin
    if(start_training ='1') then
```

```
    r_synwren <= '1';
   else if(FinishTeaching ) then
    r_synwren <= '0';
   end if;
end process SynWrEn;
```

Перед початком зміни наших синаптичних вагів, нам потрібно їх ініціалюзвати. Для цього після початку навчання ми встановлюємо це значення в '0', а після того як значення лічильника дійшло до своєї граничної межі "11", ми говоримо, що ініціалізація початковими значеннями для наших вагів відбулась.

```
Weight_Init: process(start_training, r_counter_in_i)
 begin
   if(start_training'event and start_training ='1') then
    r_weight_initialised <= '0';
   else if(r_counter_in_i = "11" ) then
    r_weight_initialised <= '1';
   end if;
end process Weight_Init;
```

В процесі SetWeightLen ми виконуємо присвоєння значення вагам за адресою. Тобто використовуючи адресу, що записана в змінній r_counter_in_i ми записуємо значення за цією адресою.

Таким чином ваги також змінюються коли змінюється лічильник. Вираз: & ”0000000” виконує заповнення нулями по причині відсутності дробової частини.

```
SetWeightLen: process (r_counter_in_i)
 begin
   if (r_counter_in_i = &amp;quot;00&amp;quot;) then
    r_synsetw <= r_decimal_weight1 & ”0000000”;
   else if (r_counter_in_i = &amp;quot;01&amp;quot;) then
    r_synsetw <= r_decimal_weight2 & ”0000000”;
   else
    r_synsetw <= r_decimal_weight3 & ”0000000”;
   end if;
end process SetWeightLen;
```

Один із важливих процесів ChrsToW виконує перетворення значення об’єкту хромосом у значення синаптичних вагів. Кожна хромосома представленя як тривимірний масив, що складається з двійкових чисел.

Після того як ми перетворили значення першої, другої та третьої хромосом на ціле число ми можемо далі використовувати їх як ваги для оптимізації значень за допомогою генетичного алгоритму.

```
ChrsToW: process(i)
  begin
    W(1):=To_integer(Signed(chrs(i)(1)));
    W(2):=To_integer(Signed(chrs(i)(2)));
    W(3):=To_integer(Signed(chrs(i)(3)));

    r_decimal_weight1 <= std_logic_vector(to_unsigned(W1,
r_decimal_weight1'length));
    r_decimal_weight2 <= std_logic_vector(to_unsigned(W2,
r_decimal_weight2'length));
    r_decimal_weight3 <= std_logic_vector(to_unsigned(W3,
r_decimal_weight3'length));
end process ChrsToW;
```

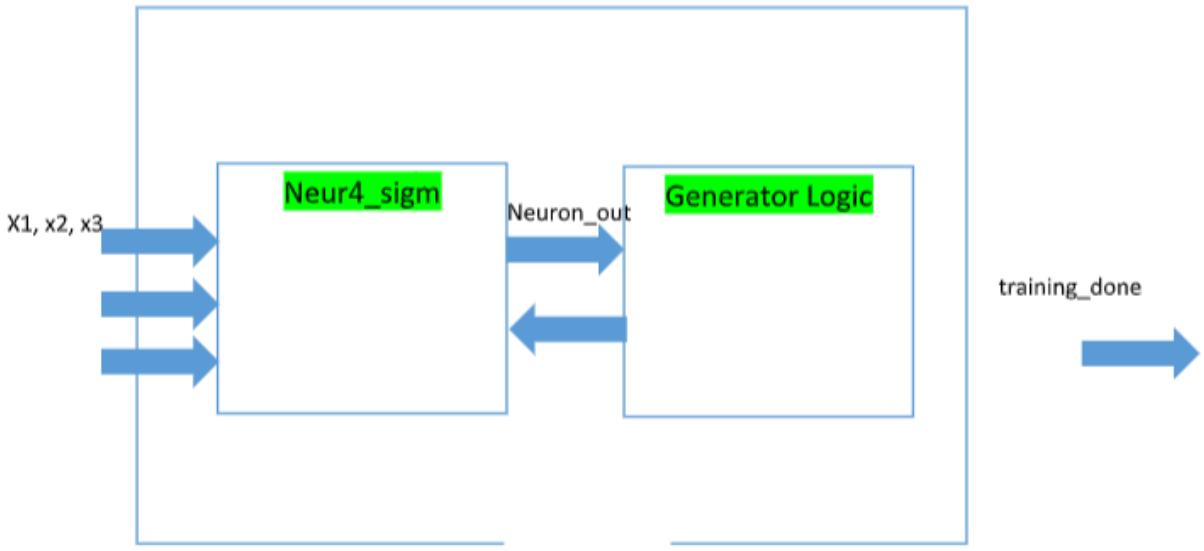

Рисунок 5.8 – структурна схема роботи модуля навчання

Виконання блоку який виконує навчання одного нейрона з чотирма входами і сигмоїдальною функцією активації можна описати такими кроками:

Крок 1. Завантажити входи x1, x2, x3 та zout до необхідного виходу. x1, x2, x3 це беззнаковий вектор довжиною 9 біт. Він представляє числа в діапазоні від 0 до 511.

Крок 2. Подати напругу(зробити значення високим) для start_training сигналу протягом 5 пікосекунд, а потім зробити низьким.

Крок 3. Чекаємо поки finish_training не набуде значення true.

Нижче зображено складніший випадок навчання, де входи та ваги було обрано випадковим чином і не підбирались спеціально для найшвидшого сходження алгоритму як це було в попередньому випадку.

На рисунку 5.10 ми бачимо як протягом кожного тіку, а саме зміни значення таймера clock_5 змінюються деякі сигнали які приймають участь у навчанні.

В нашому випадку змінна r_training_start яка встановлена в 1 показує, що процес навчання розпочався. Таким чином почала змінюватися змінна і яка позначає адресу синапсиса в який ми будемо вподальшому задавати значення вагів синапса.

Відповідно змінні r_synaddr та r_synsetw позначають адресу синапса та значення його ваг. Вони також змінюють свої значення як можна бачити в симуляції.

Перед тим як записувати ваги ми маємо явно вказати, що ми будемо виконувати запис ваг, таким чином потрібно встановити '1' в сигналі synwren.

Змінна syndaddr приймає значення "00" для перших ваг, "01" для других ваг та "10" для третіх ваг.

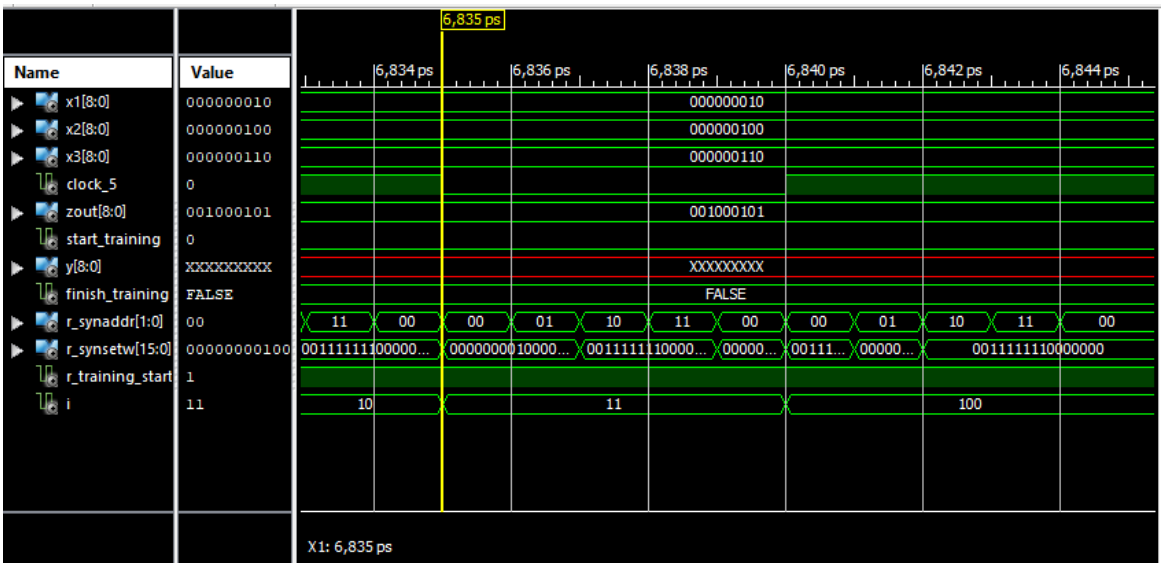

Рисунок 5.9 – симуляція навчання нейрона генетичним алгоритмом на FPGA

Для виконання симуляції було використано програмне середовище ISim(VHDL/Verilog), яке симулювало пристрій XC7A100T сімейства Artix7 із застосування пакету CSG324. Більш детально можна дізнатися на рисунку 5.10.

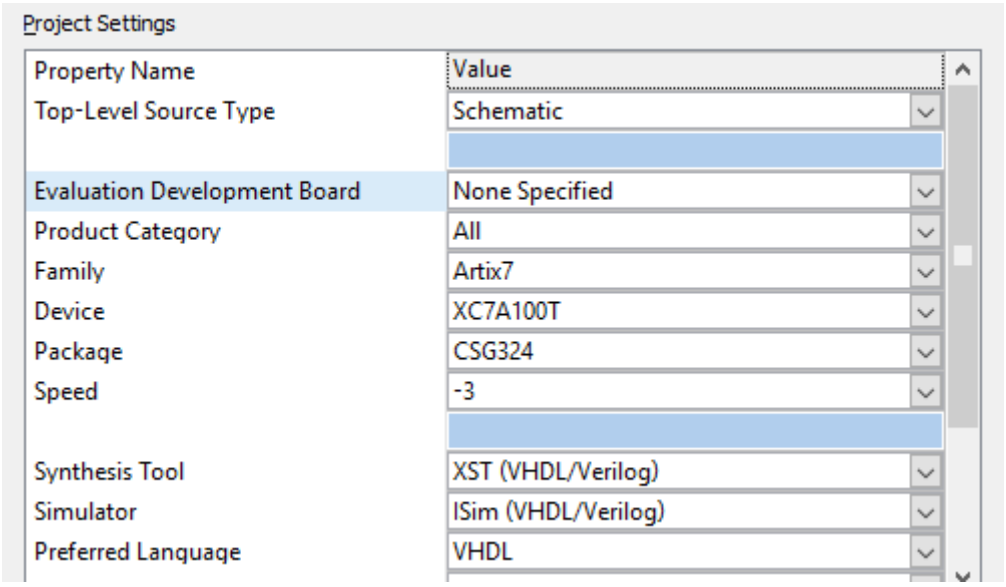

Рисунок 5.10 – характеристики симуляції для виконання генетичного алгоритму на FPGA

Щодо використаних ресурсів для виконання симуляції, то тут потрібно було 55508KB пам’яті та 1874ms процесорного часу(рисунок 5.11).

```
Time Resolution for simulation is 1ps.
Waiting for 1 sub-compilation(s) to finish...
Compiled 26 VHDL Units
Built simulation executable D:/Donwload/One_neuron/One_neuron/One_neuron/tb_neuron_with_trainer_isim_beh.exe
Fuse Memory Usage: 55508 KB
Fuse CPU Usage: 1874 ms
Launching ISim simulation engine GUI...
```

Рисунок 5.11 – характеристики використання пам’яті та процесорного часу для виконання симуляції алгоритму на FPGA

В майбутньому планується розширювати межі застосування алгоритму навчання поєднуючи блоки разом і утворюючи повнозв’язні мережі з декількома шарами/слоями як показано на рисунку 5.12.

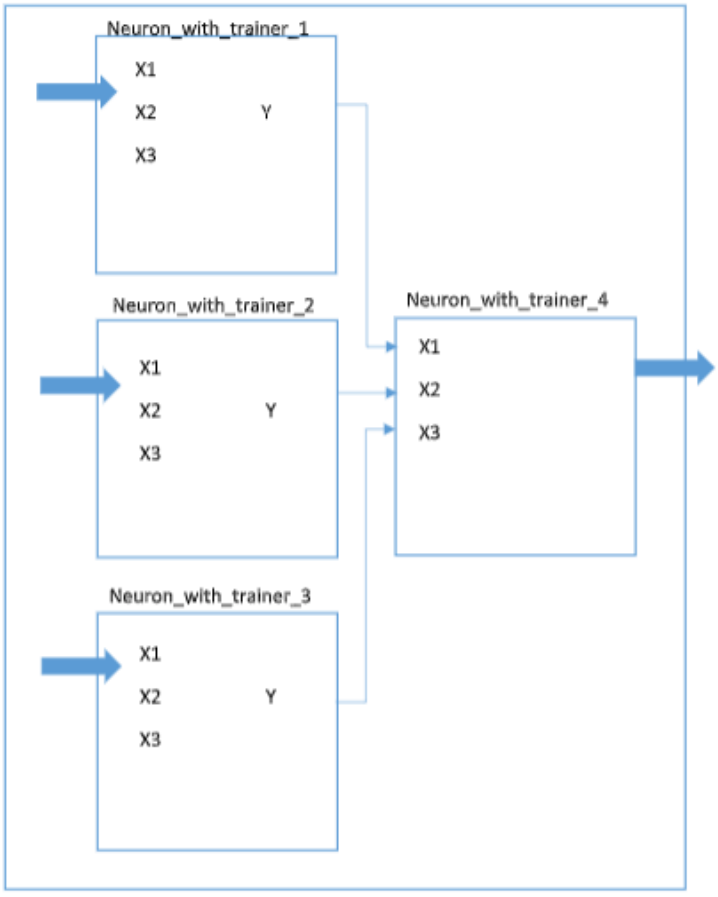

Рисунок 5.12 – Використання апаратної реалізації генетичного алгоритму для нейронних мереж з декількома шарами

Навчений за даним алгоритмом штучний нейрон з 4-ма входами і сигмоїдальною функцією активації на ПЛІС з використанням 16-розрядних чисел з фіксованою точкою зайняв 11306 LUTs (Look Up Table - вентилі логічної матриці). Алгоритм навчання для штучних нейронів розроблений на ПЛІС як окремий обчислювальній блок, зображено на рисунку 5.7. Швидкодія, як сумарна затримка комбінаційної схеми блоку нейрона, склала 75.6 нс. При зменшенні або збільшенні кількості лінійних відрізків, кількості входів нейрона або зміну розрядності чисел буде змінений і використаний ресурс ПЛІС для одного нейрона, а також точність і швидкодія.

Розглянувши розробку блоків навчання нейрона за допомогою генетичного алгоритму на ПЛІС можна винести певні результати.

Отже, під час розробки було створено працюючий прототип генетичного алгоритму навчання. За допомогою якого було навчено один нейрон.

Створення повноцінного рішення складається з блоку нейрона, блока генетичного алгоритму та інтерфейсу, що поєднує нейрон та алгоритм разом. Даний блок можна перевикористовувати в будь яких схемах, він являє собою окремий компонент.

Створено електричні схеми за допомогою мови опису апаратури VHDL та спроектовано описані блоки у середовищі ISE Design Suite 14.7.

Також було спроектовано схему на подальше використання блоку для навчання нейронних мереж з кількома слоями. Таким чином поєднуючи окремі блоки навчання і формуючи повнозв’язну мережу.

ВИСНОВКИ

В дипломному проекті вирішена науково-практична задача підвищення швидкодії навчання нейронних мереж при їх апаратній реалізації на програмованих логічних інтегральних схемах(ПЛІС) за рахунок розробки генетичного алгоритму та його апаратної реалізації на ПЛІС з використанням нейрону з чотирма входами та сигмоїдальною функцією активації.

Навчений за даним алгоритмом штучний нейрон з 4-ма входами і сигмоїдальною функцією активації на ПЛІС з використанням 16-розрядних чисел з фіксованою точкою зайняв 11306 LUTs (Look Up Table - вентилі логічної матриці). Алгоритм навчання для штучних нейронів розроблений на ПЛІС як окремий обчислювальній блок, зображено на рисунку 5.7. Швидкодія, як сумарна затримка комбінаційної схеми блоку нейрона, склала 75.6 нс. При зменшенні або збільшенні кількості лінійних відрізків, кількості входів нейрона або зміну розрядності чисел буде змінений і використаний ресурс ПЛІС для одного нейрона, а також точність і швидкодія.

Апаратна реалізація навчання нейронних мереж за допомогою генетичного алгоритму з такою швидкодією дозволяє використовувати їх в обчислювальних системах реального часу при управлінні швидкодіючими об’єктами. Зокрема, такі обчислювачі можуть застосовуватися при вирішенні задач управління автопілотом автомобіля, в безпілотній авіації. Де вони можуть вирішувати задачі розпізнавання, управління, апроксимації та класифікації отриманих даних з відеокамер та датчиків.

СПИСОК ДЖЕРЕЛ

1) Нейросетевые системы управления, Терехов В.А., Ефимов Д.В., Тюкин И.Ю. 2002. - 183с.
2) Миркес Е. М.,Нейрокомпьютер. Проект стандарта. – Новосибирск: Наука, 1999. – 337 с.
3) Анализ принципов построения и свойств устройств для моделирования нейрона, О. К. Колесницкий, к. т. н., доц.; И. В. Бокоцей; А. А. Коренной, 2012
4) http://um.co.ua/7/7-10/7-107167.html
5) http://lecture.in.ua/lekciya-2-nejronni-strukturi-v-virobnichih-tehnologiyah-civile.html
6) https://www.digikey.com/product-detail/en/xilinx-inc/XC7K160T-2FFG676C/122-1836-ND/3671575
7) https://www.digikey.com/product-detail/en/xilinx-inc/XC7V2000T-1FLG1925CES9937/122-1802-ND/3585744
8) Нейроні мережі і генетичні алгоритми – К.:«Корнійчук», . 2008. – 446 с.
9) Научный журнал – Информационно управляющие системы, 2014; Разаев Р.Р., Гоюшов А.И. – Интеллектуальная система оценки качества телекоммуникационных услуг
10) С.А. Лобов, Н.А. Сергиевский, А.А. Харламов – Адаптация алгоритма сверточных нейронных се- тей на ПЛИС, 2013
11) Krizhevsky A., Sutskever I., Hinton G. E. Imagenet classification with deep convolutional neural networks, 2012.
12) Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка, 2012: Кравець П.І., Шимкович В.М., Зубенко Г.А., Технологія апаратно-програмної реалізації штучного нейрона та штучних нейронних мереж засобами FPGA

13) Метод оптимизации весовых коэффициентов нейронных сетей с помощью генетического алгоритма при реализации на программируемых логических интегральных схемах / П.И. Кравец, В.Н. Шимкович // Электронное моделирование. — 2013. — Т. 35, № 3. — С. 65-74. — Бібліогр.: 12 назв. — рос.
14) Fletcher R., Reeves C.M. Function minimization by conjugate gradients // Computer Journal. — 1964. — Vol. 7. — P. 149—157.
15) Головко В. А. Нейронные сети: обучение, организация и применение — ИПРЖР, 2001.
16) A Genetic Algorithm Tutorial by Darrell Whitley Computer Science Department Colorado State University.
17) Pei-Yin Chen, Ren-Der Chen, Yu-Pin Chang, Leang-San Shieh, Heidar A. Malki, "Hardware Implementation for a Genetic Algorithm", Instrumentation and Measurement IEEE Transactions on, vol. 57, no. 4, pp. 699-705, 2008.
18) Genetic Algorithm Based Controls Dissertations and Theses The Stability of Genetic Algorithm Based Controllers Michael A. Marra Ph.D. Dissertation in progress.
19) Implementation of Genetic Algorithm Based on Hardware Optimization
20) Aspects of Modelling and Simulation of Genetic Algorithms: a Formal Approach
21) В.В. Гудилов, Л.А. Зинченко. Аппаратная реализация вероятностных генетических алгоритмов с параллельным формированием хромосомы
22) Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика. — 1-е. — М.: Гаряча лінія - Телекому, 2001. — С. 382.
23) В. А. Терехов, Д. В. Єфімов, И. Ю. Тюкин Нейромережні системи керування. — 1-е. — Высшая школа, 2002. — С. 184.
24) Ф. Уоссермен. Нейрокомп’ютерна техніка. Теорія і практика. — 1-е. — М.: Мир, 1992. — С. 240.

25) Саймон Хайкин. Нейроні мережі: повний курс = Neural Networks: A Comprehensive Foundation. — 2-е. — М.: «Вильямс», 2006. — С. 1104.
26) Роберт Каллан. Основні концепції нейроних мереж = The Essence of Neural Networks First Edition. — 1-е. — «Вильямс», 2001. — С. 288.
27) Л.Н. Ясницкий. Введення в штучний інтелект. — 1-е. — Издательский центр "Академия", 2005. — С. 176.
28) Г. К. Вороновский, К. В. Махотило, С. Н. Петрашев, С. А. Сергєєв. Генетичні алгоритми, штучні нейроні мережі і проблеми віртуальної реальності. — Замовне. — Х.: ОСНОВА, 1997. — С. 112
29) Д. Рутковская, М. Пилиньский, Л. Рутковский. Нейронные сети, генетические алгоритмы и нечеткие системы —1-е. — М.: Гарячая линия – Телеком, 2007. — С. 384.
30) Кравець П.І., Шимкович В.М., Ференс Д.А. Метод и алгоритмы реализации на ПЛИС функции активации для искусственных нейронных сетей. Международный научно-технический журнал Электронное моделиро-вание. – 2015. – 37, №4. – С. 63-73.
31) Petro Kravets, Volodymyr Shymkovych. Neural Network Control System with Direct and Inverse Model of Control Object Hardware and Software Realization of in FPGA/ Information Technology, Computational and Experimental Physics. Kulczycki P., Kowalski P.A., Lukasik S. (eds.) AGN-UST. - 2016. pp. 180-183
32) S.F., Telenyk, P.I., Kravets, V.M., Shymkovych, T.V., Posvistak, FPGA Implementation of the PID Algorithm for Real Time Ball Balancing on the Platform, Proceedings of The Fourth International Conference on ‘Automatic Control and Information Technology’ (ICACIT’17) December 14-16, 2017, Cracow, Poland, pp. 160-169
33) P., Kravets, V., Shymkovych, Z., Yurchenko. Algorithm for the Implementation of Radial-Basic Neural Networks and their Activation Functions on the FPGA. Proceedings of The Fourth International Conference on ‘Automatic Control and

Information Technology’ (ICACIT’17) December 14-16, 2017, Cracow, Poland, pp. 122-129